Chris Köcher

Verification of Automata with Storage Mechanisms

Verification of Automata with Storage Mechanisms

Chris Köcher

Universitätsverlag Ilmenau
2023

Impressum

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Angaben sind im Internet über http://dnb.d-nb.de abrufbar.

Diese Arbeit hat der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau als Dissertation vorgelegen

Tag der Einreichung:	01. Oktober 2021
1. Gutachter:	Prof. Dr. Dietrich Kuske (Technische Universität Ilmenau)
2. Gutachterin:	Prof. Dr. Anca Muscholl (Université Bordeaux)
3. Gutachter:	Dr. Georg Zetzsche (Max Planck Institute for Software Systems)
Tag der Verteidigung:	18. Mai 2022

Technische Universität Ilmenau/Universitätsbibliothek
Universitätsverlag Ilmenau
Postfach 10 05 65
98684 Ilmenau
https://www.tu-ilmenau.de/universitaetsverlag

ISBN 978-3-86360- 265-9 (Druckausgabe)
DOI 10.22032/dbt.53804
URN urn:nbn:de:gbv:ilm1-2022000388

Titelphoto: photocase.com

Abstract

An important research topic in computer science is the verification, i.e., the analysis of systems towards their correctness. This analysis consists of two parts: first we have to formalize the system and the desired properties. Afterwards we have to find algorithms to check whether the properties hold in the system. In many cases we can model the system as a finite automaton with a suitable storage mechanism, e.g., functional programs with recursive calls can be modeled as automata with a stack (or pushdown).

Here, we consider automata with two variations of stacks and queues:

(1) Partially lossy queues and stacks, which are allowed to forget some specified parts of their contents at any time. We are able to model unreliable systems with such memories.

(2) Distributed queues and stacks, i.e., multiple such memories with a special synchronization in between.

Often we can check the properties of our models by solving the reachability and recurrent reachability problems in our automata models. It is well-known that the decidability of these problems highly depends on the concrete data type of our automata's memory. Both problems can be solved in polynomial time for automata with one stack. In contrast, these problems are undecidable if we attach a queue or at least two stacks to our automata.

In some special cases we are still able to verify such systems. So, we will consider only special automata with multiple stacks - so-called asynchronous pushdown automata. These are multiple (local) automata each having one stack. Whenever these automata try to write something into at least one stack, we require a read action on these stacks right before these actions. We will see that the (recurrent) reachability problem is decidable for such asynchronous pushdown automata in polynomial time.

We can also semi-decide the reachability problem of our queue automata by exploration of the configration space. To this end, we can join multiple consecutive transitions to so-called meta-transformations and simulate them at once. Here, we study meta-transformations alternating between writing words from a given regular language into the queues and reading words from another regular language from the queues. We will see that such meta-transformations can be applied in polynomial time.

We will prove the aforementioned results with the help of several algebraic properties of the storage mechanisms. To this end, we will first study the monoid of all action sequences which can be applied on such memories. This monoid is also known as the *transformation monoid* or *action monoid*. So, we will see for each of our considered storage mechanisms that for each sequence of basic actions we can construct in any case another action sequence which is somewhat "simple" and has the same effect as the original sequence. Furthermore, we will characterize several classes of languages in these transformation monoids and analyze their algorithmic properties.

Zusammenfassung

Ein wichtiges Forschungsthema in der Informatik ist die Verifikation, d.h., die Analyse von Systemen bezüglich ihrer Korrektheit. Diese Analyse erfolgt in zwei Schritten: Zuerst müssen wir das System und die gewünschten Eigenschaften formalisieren. Anschließend benötigen wir Algorithmen zum Testen, ob das System die Eigenschaften erfüllt. Oftmals können wir das Systemals endlichen Automaten mit geeignetem Speichermechanismus modellieren, z.B. rekursive Programme sind im Wesentlichen Automaten mit einem Stack.

Hier betrachten wir Automaten mit zwei Varianten von Stacks und Queues:

(1) Partiell vergessliche Stacks und Queues, welche bestimmte Teile ihrer Inhalte jederzeit vergessen können. Diese können für unzuverlässige Systeme verwendet werden.

(2) Verteilte Stacks und Queues, d.h., mehrere Stacks und Queues mit vordefinierter Synchronisierung.

Häufig lassen sich die Eigenschaften unserer Modelle mithilfe des (wiederholten) Erreichbarkeitsproblems in unseren Automaten lösen. Dabei ist bekannt, dass die Entscheidbarkeit dieser Probleme oftmals stark vom konkreten Datentyp des Speichers abhängt. Beide Probleme können für Automaten mit einem Stack in Polynomialzeit gelöst werden. Sie sind jedoch unentscheidbar, wenn wir Automaten mit einer Queue oder zwei Stacks betrachten.

In bestimmten Spezialfällen sind aber dennoch in der Lage diese Systeme zu verifizieren. So können wir beispielsweise bestimmte Automaten mit mehreren Stacks betrachten - so genannte Asynchrone Kellerautomaten. Diese bestehen aus mehreren (lokalen) Automaten mit jeweils einem Stack. Wann immer diese Automaten etwas in mind. einen Stack schreiben, müssen sie unmittelbar zuvor von diesen Stacks etwas lesen. Das (wiederholte) Erreichbarkeitsproblem ist in asynchronen Kellerautomaten in Polynomialzeit entscheidbar.

Wir können zudem das Erreichbarkeitsproblem von Queueautomaten durch Exploration des Konfigurationsraums semi-entscheiden. Hierzu können wir mehrere aufeinanderfolgende Transitionen zu so genannten Meta-Transformationen zusammenfassen und diese in einem Schritt simulieren. Hier betrachten wir Meta-Transformationen, die zwischen dem Lesen und Schreiben von Wörtern aus zwei gegebenen regulären Sprachen alternieren. Diese Meta-Transformationen können in Polynomialzeit ausgeführt werden.

Wir verwenden hierfür verschiedene algebraische Eigenschaften, die wir zuvor in dieser Arbeit erlangen werden. Konkret, betrachten wir das Monoid bestehend aus allen (verschiedenen) Aktionsfolgen auf einem gegebenen Datentyp - das sogenannte *Transformationsmonoid*. Unter anderem werden wir sehen, dass in jedem unserer betrachteten Datentypen für jede Aktionsfolge eine weitere solche Folge berechnet werden kann, die auf gewisse Art und Weise „einfach“ ist und dabei denselben Effekt besitzt wie die ursprüngiche Aktionsfolge. Zudem charakterisieren wir verschiedene Klassen von Teilmengen der Transformationsmonoide und untersuchen deren algorithmische Eigenschaften.

Danksagung

Nach einigen Jahren der harten Arbeit, konnte ich diese Dissertation endlich fertiggestellen. Dies geschah jedoch nicht ohne die Unterstützung einiger besonderer Menschen in meinem Leben. Es ist deswegen an der Zeit, mich an dieser Stelle bei all diesen Menschen zu bedanken:

Als erstes möchte ich mich hier natürlich bei dem Betreuer meiner Arbeit, Prof. Dietrich Kuske, bedanken. Er hatte nahezu immer ein offenes Ohr, wenn ich in meiner Forschung an besonders schwierigen Stellen einmal nicht weiter wusste, und tat sein Bestes, um mir dabei auch weiter zu helfen. Insbesondere aber nahm er sich auch stets die Zeit, meine Entwürfe - sowohl für diese Arbeit wie auch anderer Publikationen - gegenzulesen. Dabei profitierten diese Arbeiten ungemein durch seine zahllosen hilfreichen Vorschläge, wie ich noch Beweise vereinfachen könnte und damit die Arbeiten an sich auch verständlicher machen konnte.

Ich möchte mich ebenso auch bei den beiden externen Gutachter/innen dieser Arbeit, Prof. Anca Muscholl und Dr. Georg Zetzsche, bedanken, dass sie sich für das Lesen und Auswerten dieser Arbeit so viel Zeit genommen haben. Mit Georg habe ich mich auch bereits zuvor auf einigen Konferenzen getroffen und über meine Forschung gesprochen. Ich möchte mich deshalb bei ihm auch für seinen zusätzlichen Input bedanken.

Des Weiteren gilt mein Dank auch den Kolleg/innen vom Institut für Theoretische Informatik, namentlich (in alphabetischer Reihenfolge): Faried Abu Zaid, Arindam Biswas, Christopher Hugenroth, Olena Pryanychnykova, Philipp Schlag, Christian Schwarz und Stefan Walzer. Mit ihnen konnte ich auch (fast) jederzeit über die Arbeit reden, aber auch über Gott und die Welt, wenn ich einmal ein wenig Ablenkung benötigte, um wieder einen klaren Kopf zu bekommen. Zu Zweiterem hat insbesondere der bis vor Lockdown-Zeiten regelmäßig stattfindende Spieleabend beigetragen (an dieser Stelle sei dann auch den externen Spieleabend-Teilnehmern Sascha Grau und Adrian Grewe gedankt). Ich danke den Kollegen außerdem für den selbstgebastelten Doktorhut.

Gemeinsam mit Olena und Faried sind auch zwei Publikationen entstanden, die ich so im Alleingang wohl niemals geschafft hätte. Ein besonderer Dank gilt aber auch Philipp, der in dieser Zeit zu einem sehr guten Freund geworden ist, mit dem ich auch außerhalb der Universität viel gemeinsame Zeit mit Filmabenden und Urlaubsreisen verbracht habe. Er hat auch eine ganze Menge Freizeit aufgewendet, sich diese Arbeit komplett durchzulesen und Verbesserungsvorschläge zu unterbreiten. Hierfür danke ich ihm besonders.

Nicht zuletzt gebührt noch ein besonderer Dank meinem Freundeskreis, insbesondere Andreas Neumann, Stefan Nöth, Isabell Scholz, Sebastian Themel, Felix Wiemuth und Sabine Zorn, sowie meiner gesamten Familie für ihre seelische und moralische Unterstützung über all die Jahr hinweg.

Contents

Chapter 1
Introduction

1.1 Motivation and Historical Context

One of the most important questions in computer science is whether a given program or system is correct. In this connection, "correct" means that the program or system conforms to a given specification. In automata theory one tackles this question as follows: first we seek a mathematical model of such program or system. In many cases, an appropriate model is a finite state machine (or automaton) with a suitable storage mechanism. For example, with the help of a finite-state automaton with a finite memory we can model the principal control-flow in imperative programs (with finite domains) or in distributed and concurrent systems. Note that such automata are essentially *(non-)deterministic finite automata* (NFA resp. DFA, for short). If we attach a stack we obtain so-called *pushdown automata* (PDA, for short). With the help of such PDAs we can model functional programs with finite domains in which we are allowed to call functions recursively. If we replace the stack by several queues, we obtain (multi-)queue automata and can model finite computer networks communicating through channels.

After modeling our system as an automaton with storage, we also have to formalize the expected behavior of the system with the help of special logics, so-called *temporal logics*. With the help of those logics we are able to reason about properties qualified in terms of time. The most important of such logics are *propositional* or *linear temporal logic* (PTL or LTL), *computational tree logic* (CTL), and their combination CTL*.

Finally, we have to check whether our system conforms to the expected behavior. To this end, we check whether the formula modeling the system's behavior is satisfied by the automaton modeling the system. In other words, we have to solve the *model checking problem* of temporal logics in automata with storage mechanisms. In many cases, this problem reduces to the *recurrent reachability problem* of our automata model. This problem asks whether a given automaton has an infinite run visiting one or more given configurations infinitely often.

We already know that the decidability of the recurrent reachability problem (and even the "simple" *reachability problem*) highly depends on the data type of the memory attached to our automata. So, the reachability problem and the recurrent reachability problem in NFAs both are decidable using nondeterministically logarithmic space, only. Additionally, the model checking problem of LTL- and CTL*-formulas is PSPACE-complete [Pnu77, VW86] in this case while model checking of CTL-formulas is possible in polynomial time [CE82].

For pushdown automata, reachability and recurrent reachability are decidable in polynomial time [BEM97, FWW97]. Their LTL- and CTL-model checking problem is complete in the complexity class EXPTIME [BEM97, Wal00, Wal01] and their model checking for

CTL*-formulas is 2EXPTIME-complete [Boz07]. Interestingly, a finite automaton with more than one stack is as powerful as a Turing-machine resulting in the undecidability of any non-trivial verification problem.

We can also consider automata with counters, so-called *counter automata*. Note that there are multiple types of counters: a blind counter is essentially an integer which can be increased or decreased. Partially blind counters are defined similarly, but their decrease operation fails if the stored number falls below zero. A counter with zero-test is a partially blind counter which can additionally check, whether its content is zero. Anyway, we can model an automaton with a single counter as a pushdown automaton with some restrictions to its stack. However, our automata are able to encode Turing-machines if we attach at least two counters with zero-tests [Min67]. We also call such automata *Minsky-machines*. The reachability problem is decidable if we replace the counters by partially blind ones [May84]. Unfortunately, this decision problem has an Ackermannian complexity as Czerwiński and Orlikowski as well as Leroux independently proved recently in their papers [CO22, Ler22]. Note that in literature such automata are also known as *vector addition systems* (VAS or VASS, for short) or *Petri nets*.

We also know that queue automata (even with only one queue) are as powerful as Turing-machines (cf. [BZ83]). Hence, reachability and model checking of temporal logics is undecidable. But if we allow the automaton's queue(s) to forget some letters (in this case we call them *lossy queue automata*), the reachability problem gets decidable [AJ96a]. Note that this problem has a very high complexity [Sch02, CS08]. However, model checking of temporal logics is undecidable [AJ96b]. The decidability of the reachability problem can also be extended to so-called *well-structured transition systems* by [Abd+00, FS01].

Another variation of queues are so-called priority queues. Here, any queue entry has a priority with the following semantics: entries with a high priority can supersede or overtake entries with lower priority. Haase et al. proved in [HSS14] that reachability is decidable for priority queue automata with superseding semantics, but undecidable for those with overtaking semantics.

In this thesis we want to direct the focus on two further variations of queues and stacks. The first considered variation are the so-called *partially lossy* queues and stacks. We can see this as a kind of unification of classical reliable and lossy memories. Concretely, in this case we specify two alphabets F and U where F (the *forgettable* letters) specifies the entries which can be forgotten at any time and U (the *unforgettable* letters) specifies the entries which cannot get lost. With the help of these special storage mechanisms we are able to prove our contributions for reliable and lossy queues or stacks, respectively, at the same time. We will see in this paper that automata having a partially lossy queue as their memory are still as powerful as Turing-machines - at least if there is one or more unforgettable letter. In contrast, automata with a partially lossy stack are just special pushdown automata.

The second considered variation are distributed queues and stacks. These consist of a finite number of reliable queues or stacks with a special synchronization mechanism between them. We can also see these storage mechanisms as queues and stacks containing (Mazurkiewicz) traces instead of words (for an overview on the trace theory see, e.g., [Maz77, DR95]). With the help of such memories we are able to model computer networks where each computer is a queue or pushdown automaton. In the end, we can see distributed queue or pushdown automata as a generalization of the classical multi-queue or multi-stack automata as well as a generalization of communicating automata (cf. [KM21] for an overview

on this topic). Anyway, distributed queue and pushdown automata are Turing-complete and, hence, have an undecidable reachability problem.

As we have noted, in the most kinds of automata the reachability problem is either undecidable or very inefficient. Hence, the analysis of those automata requires approximations of this problem. A first, very trivial approach is to under-approximate the set of reachable configurations step-by-step. In other words, from the set of configurations which are reachable via $n \in \mathbb{N}$ computation steps, we easily obtain the configurations reachable after $n + 1$ steps. Even if we are able to semi-decide the reachability problem via this algorithm, it is not very efficient. So, after n computation steps we have still explored only a finite extension of the configuration space.

A more efficient under-approximation was introduced by Boigelot et al. in [Boi+97] for reliable queue automata and was translated to lossy queue automata by Abdulla et al. in [Abd+04]. Here, we combine multiple computation steps of the automaton to a so-called *meta-transformation*. In particular, in the named papers the authors considered the set of configurations reachable via looping through a finite sequence of basic queue actions. If one starts from a regular language of configurations the set of reachable configurations is still regular in this case. So, in the aforementioned algorithm, we additionally search for such loops and apply them at once. With this modification of the trivial algorithm we are able to explore an infinite extension of the set of considered initial configurations in just a small amount of time.

Another possibility to partially solve the reachability problem is to restrict the finite-state automata. One example of such automata are so-called *flat automata*. In these automata any control state belongs to at most one loop, i.e., there are no nested or interleaving loops. The reachability problem for such flat (lossy or reliable) queue automata is NP-complete [FP20, Sch20]. We can prove this result by utilization of the approach from Boigelot et al. respectively from Abdulla et al. This can be generalized to so-called *input-bounded automata*. In this case, the write actions to be applied to the attached queue are finite products of loops and words. Such queue automata reduce to multi-counter automata without zero-tests by [BFS20]. Hence, this reduction yields an algorithm deciding the reachability problem in non-primitive recursive complexity. However, it is unknown if it also shares the lower complexity bound of reachability in multi-counter automata.

We will see in this thesis just another approach towards a decision procedure of the reachability problem. So we will consider distributed pushdown automata in which we restrict the actions the automaton can apply to its stacks. Concretely, we allow the application of write actions only on those stacks from which we have read a letter right before the write actions. Such behavior prevents that we are able to copy data from one stack to another one. In this case, we will obtain an efficient algorithm deciding the reachability problem of those restricted distributed pushdown automata. This also yields the decidability of a trace-like variation of the temporal logic LTL.

1.2 Outline and Main Contributions

The main part of this thesis is divided into two big parts. Roughly, in the first part we study the action sequences a finite automaton is allowed to apply to its attached queue or stack. In the second part we will use the knowledge we have learned before and analyze the reachability problem and the model checking problem of temporal logics on those automata.

Right before the first part, we define a few basic definitions in Chapter 2, which we use across the whole thesis. Additionally, in Chapter 3 we formalize storage mechanisms with the help of *data types*. We also present a couple of basic data types as well as the already mentioned partially lossy and distributed variants of queues and stacks.

1.2.1 Properties of Transformation Monoids

In the first part of this thesis we study the transformations that can be applied in queue and pushdown automata. These transformations form a special monoid, the so-called *transformation monoid* or *action monoid*. This monoid is a quotient of the free monoid on the sequences of basic actions of the studied data type.

In Chapter 4 we study some basic properties of the transformation monoid of partially lossy queues and stacks. Concretely, we define the so-called *behavioral equivalence* which identifies the sequences of actions which always have the same effect on any content of the considered data type. This equivalence is an important tool when studying the behavior of a data type and verifying automata having a memory of such data type. In this chapter we list several equations which fully characterize this behavioral equivalence and, therefore, the transformation monoid. If we understand those equations as a semi-Thue system we obtain for any sequence of actions a unique, irreducible word which also is in some sense "simple". Afterwards, we study the shape of the normal forms when concatenating two transformation sequences. We also characterize Green's relations and cancellation properties of the considered monoids. Finally, we also recall the embedding properties which we stated in [Köc16, KKP18].

In the succeeding chapter we consider the algorithmic properties of rational languages in the transformation monoids. These are exactly those transformations an automaton is allowed to apply to its attached memory. For the partially lossy queues we will see that the rational membership problem is decidable using only nondeterministic logarithmic space, but other problems like equivalence, universality, or intersection emptiness are undecidable. We also show a characterization of the rational languages in the transformation monoid of partially lossy stacks. Concretely, a language is rational in this monoid if, and only if, the language of the contained normal forms (from the previous chapter) is regular. This characterization is a generalization of a result by Render and Kambites [RK09]. From this result we also obtain the decidability of almost all of the considered decision problems.

Due to the undecidability of the considered decision problems for the rational languages in the transformation monoid of partially lossy queues, we also consider a proper subclass: the *recognizable* languages. These are those regular languages of action sequences which are closed under behavioral equivalence. Using the well-known algorithms from the automata theory, we obtain the decidability of almost all of the considered problems. In Chapter 6 we finally characterize in which cases a language of action sequences is recognizable. To this end, we give a characterization in terms of Kleene's Theorem [Kle51] and in terms of Büchi's Theorem [Büc60]. In other words, we define special rational expressions generating exactly the recognizable languages and we define a monadic second-order logic describing this class of languages. Afterwards, we also prove that partially lossy queues have (in most cases) only trivial recognizable languages: the empty set and the transformation monoid itself.

In the last chapter of the first part, we finally consider the aperiodic languages in the transformation monoid. These are those recognizable languages in which we cannot count modulo any natural number. For partially lossy queues we give characterizations in the

style of Schützenberger [Sch65] and McNaughton-Papert [MP71]. In other words, we give a restriction to the star-free expressions generating the aperiodic languages. Similarly, the first-order fragment of our special MSO-logic describing the recognizable languages, also describes the aperiodic languages in this monoid.

1.2.2 Reachability Problems and Verification

In the second part we will finally utilize our knowledge about the transformation monoid when verifying finite automata with our special variations of stacks and queues.

First, in Chapter 8 we recall some basic definitions and results we need for verification of queues and stacks. Concretely, we define several reachability problems as well as several temporal logics like LTL, CTL, and CTL*. We also recall the approach from Vardi and Wolper to prove the decidability of the model checking problem of formulas from several of the considered logics [VW86].

In Chapter 9 we study the reachability and model checking problem for the aforementioned variations of stacks. We start with recalling the constructions from [BEM97, FWW97] considering the reachability and recurrent reachability problem of classical pushdown automata. We can also transfer this result to partially lossy stacks. In this case we show that a partially lossy PDA is essentially a classical one with some additional loops on any control state. Finally, we consider distributed stacks. Since automata having at least two stacks are already Turing-powerful, we restrict the transitions of the considered automata as we have mentioned before. In those restricted, distributed PDAs (so-called *asynchronous* PDAs), we show that the set of backwards reachable configurations are efficiently recognizable if we start from given recognizable set of configurations. This construction is close to the one from Bouajjani et al. in [BEM97], but even more sophisticated. This result also yields efficient algorithms deciding the reachability and the recurrent reachability problem. We have also considered the forwards reachable configurations in an asynchronous pushdown automaton. We will see that the set of configurations forwards reachable from a given rational set of configurations always is rational. To this end, we will see that each forwards reachable configuration also is reachable via a special, "short" run of our asynchronous automaton. For a fixed underlying alphabet our construction requires only polynomial time.

Finally, in Chapter 10 we consider the reachability problem of reliable, partially lossy, and distributed queues. Concretely, we want to extend the approach from Boigelot et al. and Abdulla et al. in [Boi+97, Abd+04]. To this end, we introduce several further kinds of meta-transformations. We will see that we reach a regular set of configurations from another one by application of recognizable languages in the transformation monoid (see Chapter 6). Additionally, we introduce so-called *read-write independent* sets. These are essentially languages alternating between a regular language of write action sequences and a regular language of read action sequences. For these sets we also prove that the set of reachable configurations is regular if we start from another regular set of configurations. We prove this result by a tricky reduction to the reachability problem of a classical pushdown automaton (or actually a 1-counter automaton without zero-tests). We also see that all of our constructions are possible in polynomial time (at least for a fixed underlying alphabet).

Chapter 2
Preliminaries

In this chapter we recall some basic definitions and notions which we require all across this thesis.

2.1 Functions and Relations

Let $f\colon I \to O$ be a function with pre-image I and image O. For $i \in I$ and $o \in O$ we write $f[i \mapsto o]$ for the function $g\colon I \to O$ with $g(i) = o$ and $g(x) = f(x)$ for each $x \in I \setminus \{i\}$. In other words, $f[i \mapsto o]$ is f in which we replace the image of i by o.

Let $R \subseteq S \times T$ be a binary relation on S and T. For $s \in S$ and $t \in T$ we also write $s\,R\,t$ instead of $(s,t) \in R$. We also define the sets $s\,R := \{t \in T \mid s\,R\,t\}$ and $R\,t := \{s \in S \mid s\,R\,t\}$. We can also extend these notions to subsets of S and T: $S'\,R := \bigcup_{s \in S'} s\,R$ and $R\,T' := \bigcup_{t \in T'} R\,t$.

Now, let $R = \prod_{i=1}^{n} S_i$ be a finite product of sets. For $I \subseteq \{1,\dots,n\}$ we write $R{\restriction_I} := \prod_{i \in I} S_i$ for the projection (or restriction) of R to components with indices in I. Similarly, for $\vec{s} = (s_i)_{1 \le i \le n} \in R$ we write $\vec{s}{\restriction_I} = (s_i)_{i \in I}$. Let $I, J \subseteq \{1,\dots,n\}$ be two disjoint sets of indices and $\vec{s} \in R{\restriction_I}$ and $\vec{t} \in R{\restriction_J}$ be two elements. Then by $(\vec{s},\vec{t})$ we denote the joint tuple $\vec{r} \in R{\restriction_{I \cup J}}$ with $\vec{r}{\restriction_I} = \vec{s}$ and $\vec{r}{\restriction_J} = \vec{t}$.

Let $\le$ be a quasi ordering on a set S and $T \subseteq S$. The *downwards* and *upwards closure* of T wrt. $\le$ are the sets

$$\downarrow_{\le} T := \{s \in S \mid \exists t \in T\colon s \le t\} \quad \text{and} \quad \uparrow_{\le} T := \{s \in S \mid \exists t \in T\colon t \le s\}\,.$$

The set T is *downwards closed* (*upwards closed*) wrt. $\le$ if we have $T = \downarrow_{\le} T$ ($T = \uparrow_{\le} T$, resp.).

Now, let $\equiv$ be an equivalence relation on S and $T \subseteq S$. Then the *closure* of T wrt. $\equiv$ is the set

$$[T]_{\equiv} := \{s \in S \mid \exists t \in T\colon s \equiv t\}\,.$$

A set T is *closed* under $\equiv$ if we have $T = [T]_{\equiv}$.

2.2 Monoids

A *monoid* is a tuple $(\mathbb{M}, \cdot, e)$ where $\mathbb{M}$ is a set, $\cdot\colon \mathbb{M}^2 \to \mathbb{M}$ is an associative operation on $\mathbb{M}$, and $e \in \mathbb{M}$ is the *identity* satisfying $m \cdot e = e \cdot m = m$ for each $m \in \mathbb{M}$. Note that a monoid contains exactly one element satisfying this equation. Whenever the situation is clear, we write $\mathbb{M}$ instead of $(\mathbb{M}, \cdot, e)$. Let $m, n \in \mathbb{M}$ with $mn = e$. Then m is called a *left-inverse* of n while n is called a *right-inverse* of m. Note that in general not all elements of a monoid have a left- or right-inverse. Additionally, there may be elements having more than one left- resp.

right-inverses. A monoid is called a *group* if all elements have a unique left- and right-inverse. In this case the left- and right-inverse element of any $m \in \mathbb{M}$ coincide. We call this element the *inverse* element of m and denote it by m^{-1} (or in some contexts $-m$).

Example 2.2.1. (1) The *natural numbers* $\mathbb{N} := \{0, 1, 2, \dots\}$ with addition + is a monoid with identity 0. If we extend the natural numbers by the negative numbers, we obtain the set of *integers* $\mathbb{Z} := \{\dots, -2, -1\} \cup \mathbb{N}$. This set with addition is even a group.

(2) Let Γ be an *alphabet*, i.e., Γ is a finite, non-empty set. Then the set $\Gamma^* := \{a_1 a_2 \dots a_n \mid n \in \mathbb{N}, a_1, a_2, \dots, a_n \in \Gamma\}$ with the concatenation $\cdot$ is a monoid - the so-called *free monoid* over Γ. Its identity ε is the word of length 0 (or the *empty word*). Elements $w \in \Gamma^*$ are also called *words* and subsets $L \subseteq \Gamma^*$ are called *(word) languages*. ⌟

A monoid $\mathbb{M}$ is *left-cancellative* if for each $k, m, n \in \mathbb{M}$ the following implication holds:

$$km = kn \implies m = n\,.$$

Similarly, $\mathbb{M}$ is *right-cancellative* if for each $k, m, n \in \mathbb{M}$ the following implication holds:

$$mk = nk \implies m = n\,.$$

$\mathbb{M}$ is *cancellative* if it is both, left- and right-cancellative. For example, Γ^*, $\mathbb{N}$, and $\mathbb{Z}$ are cancellative. We call $m \in \mathbb{M}$ a *prefix* (*suffix*) of $k \in \mathbb{M}$ if we have $k = mn$ ($k = nm$, resp.) for an $n \in \mathbb{M}$. If $\mathbb{M}$ is left-cancellative (right-cancellative, resp.) the element n is uniquely defined and is called the *complementary suffix* (*prefix*, resp.) of k wrt. m. In literature, the element m is also called a *left- or right-divisor* of k and n is the *left- or right-quotient* of k wrt. m. The element m is an *infix* (or *factor*) of k if there are elements $\ell, n \in \mathbb{M}$ with $k = \ell m n$.

Now, let Γ be an alphabet. For $w = \alpha_1 \dots \alpha_n$ (with $\alpha_1, \dots, \alpha_n \in \Gamma$), $1 \le i \le n$, and $0 \le j \le n$ we denote $w[i, j] := \alpha_i \dots \alpha_j$ the infix of w from position i to j. We assume $w[i, j] = \varepsilon$ if $i > j$ holds.

In algebra, *Green's relations* are an important concept. These are defined as follows: let $\mathbb{M}$ be a monoid and $m, n \in \mathbb{M}$. We write

$$\begin{aligned}
m \,\mathrm{L}\, n &\iff \mathbb{M}m = \mathbb{M}n,\\
m \,\mathrm{R}\, n &\iff m\mathbb{M} = n\mathbb{M},\\
m \,\mathrm{J}\, n &\iff \mathbb{M}m\mathbb{M} = \mathbb{M}n\mathbb{M},\\
m \,\mathrm{H}\, n &\iff m \,\mathrm{L}\, n \text{ and } m \,\mathrm{R}\, n,\\
m \,\mathrm{D}\, n &\iff \exists k \in \mathbb{M}\colon m \,\mathrm{L}\, k \text{ and } k \,\mathrm{R}\, n\,.
\end{aligned}$$

Note that L, R, H, and D are contained in the relation J.

2.3 Graphs

A *(directed) graph* is a tuple $G = (V, E)$ where V is a set of *vertices* or *nodes* and $E \subseteq V^2$ is a set of *edges*. For two nodes $u, v \in V$ we write $u \to_G v$ if $(u, v) \in E$. For $u, v \in V$ we write $u \to_G^* v$ if there is a sequence $v_0, v_1, \dots, v_n \in V$ with $v_0 = u$, $v_{i-1} \to_G v_i$ for each $1 \le i \le n$, and $v_n = v$. Such sequence is also called a *path* in G. Note that $\to_G^*$ is a reflexive and transitive relation. For $I, F \subseteq V$ we write $I \to_G^* F$ if there are $\iota \in I$ and $f \in F$ with $\iota \to_G^* f$. Additionally,

we write $I \to_{\mathcal{G}}^* v$ and $v \to_{\mathcal{G}}^* F$ instead of $I \to_{\mathcal{G}}^* \{v\}$ resp. $\{v\} \to_{\mathcal{G}}^* F$ (where $v \in V$ is a node). A set $U \subseteq V$ is *(weakly) connected* if for each $u, v \in U$ we have $u \to_{\mathcal{H}}^* v$ where $\mathcal{H} = (V, F)$ is the symmetric closure of $\mathcal{G}$, i.e., we have $F = E \cup \{(u, v) \in V^2 \mid (v, u) \in E\}$. A set $U \subseteq V$ is a *connected component* if it is connected and for each $U' \supsetneq U$ the set U' is not connected, i.e., U is a maximal connected set. The *induced subgraph* $\mathcal{G}|_U := (U, E \cap U^2)$ is the $\mathcal{G}$ restricted to the nodes in $U \subseteq V$. The *complementary graph* of $\mathcal{G}$ is $\mathcal{G}^c := (V, V^2 \setminus E)$.

Now, let Γ be a non-empty set. A *Γ-labeled graph* is a tuple $\mathcal{G} = (V, E)$ where V is a set of vertices and $E \subseteq V \times \Gamma \times V$ is a set of (labeled) edges. For $\alpha \in \Gamma$, $u, v \in V$ we write $u \xrightarrow{\alpha}_{\mathcal{G}} v$ if $(u, \alpha, v) \in E$. For a sequence $\gamma \in \Gamma^*$ and vertices $u, v \in V$ we write $u \xrightarrow{\gamma}_{\mathcal{G}} v$ if there are $\alpha_1, \ldots, \alpha_n \in \Gamma$ and $v_0, v_1, \ldots, v_n \in V$ with $\gamma = \alpha_1 \ldots \alpha_n$, $v_0 = u$, $v_{i-1} \xrightarrow{\alpha_i}_{\mathcal{G}} v_i$ for each $1 \leq i \leq n$, and $v_n = v$. In this case, we call the sequence $v_0, \ldots, v_n$ a γ-labeled path. For $I, F \subseteq V$ and $\gamma \in \Gamma^*$ we write $I \xrightarrow{\gamma}_{\mathcal{G}} F$ if there are $\iota \in I$ and $f \in F$ with $\iota \xrightarrow{\gamma}_{\mathcal{G}} f$. Similarly, we drop the braces if I or F is a singleton.

2.4 Automata and Languages

A *nondeterministic finite automaton* (NFA, for short) is a tuple $\mathfrak{A} = (Q, \Gamma, I, \Delta, F)$ where Q is a finite set of *states*, Γ is an alphabet, $I, F \subseteq Q$ are the sets of *initial* and *final states*, respectively, and $\Delta \subseteq Q \times \Gamma \times Q$ is a set of *transitions*. The *configuration graph* of $\mathfrak{A}$ is the Γ-labeled graph $\mathcal{G}_{\mathfrak{A}} = (Q, \Delta)$. For $p, q \in Q$ and $w \in \Gamma^*$ we also write $p \xrightarrow{w}_{\mathfrak{A}} q$ instead of $p \xrightarrow{w}_{\mathcal{G}_{\mathfrak{A}}} q$. A w-labeled path in $\mathcal{G}_{\mathfrak{A}}$ is also called a *run* of $\mathfrak{A}$ with label w. The *accepted language* of $\mathfrak{A}$ is $L(\mathfrak{A}) := \{w \in \Gamma^* \mid I \xrightarrow{w}_{\mathfrak{A}} F\}$. A language $L \subseteq \Gamma^*$ is called *regular* if there is an NFA $\mathfrak{A}$ with $L = L(\mathfrak{A})$. For a pair $P_1, P_2 \subseteq Q$ we write $\mathfrak{A}_{P_1 \to P_2}$ for the NFA $(Q, \Gamma, P_1, \Delta, P_2)$, i.e., we modify the initial and final states of NFA to P_1 and P_2, respectively. Then we have the following equation:

$$L(\mathfrak{A}) = \bigcup_{q \in Q} L(\mathfrak{A}_{I \to q}) \cdot L(\mathfrak{A}_{q \to F}) \,.$$

We can also extend the notion of NFAs as follows: an (extended) NFA is a quintuple $\mathfrak{A} = (Q, \Gamma, I, \Delta, F)$ where Q, Γ, I, and F are defined as before and $\Delta \subseteq Q \times \Gamma^* \times Q$ is finite. We already know that, from a given extended NFA $\mathfrak{A}$, we can construct a (classical) NFA accepting $L(\mathfrak{A})$.

An NFA $\mathfrak{A} = (Q, \Gamma, I, \Delta, F)$ is called *deterministic* (or a DFA) if we have $|I| = 1$ and for each $p \in Q$ and $\alpha \in \Gamma$ there is at most one $q \in Q$ with $(p, \alpha, q) \in \Delta$. DFAs also accept exactly the class of regular languages.

Let Γ be an alphabet and $K, L \subseteq \Gamma^*$ be two languages. The *shuffle* of K and L is the language

$$K \sqcup\!\sqcup L := \left\{ v_1 w_1 v_2 w_2 \ldots v_n w_n \;\middle|\; \begin{array}{c} n \in \mathbb{N}, v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_n \in \Gamma^*, \\ v_1 v_2 \ldots v_n \in K, w_1 w_2 \ldots w_n \in L \end{array} \right\} .$$

Now, let $\Sigma \subseteq \Gamma$. The *projection* to Σ is the homomorphism $\pi_\Sigma \colon \Gamma^* \to \Sigma^*$ with

$$\pi_\Sigma(\varepsilon) = \varepsilon \quad \text{and} \quad \pi_\Sigma(\alpha w) = \begin{cases} \alpha \pi_\Sigma(w) & \text{if } \alpha \in \Sigma \\ \pi_\Sigma(w) & \text{if } \alpha \notin \Sigma \end{cases}$$

for each $\alpha \in \Gamma$ and $w \in \Gamma^*$. The *reversal* of a word $w = \alpha_1 \alpha_2 \ldots \alpha_n$ (with $\alpha_1, \alpha_2, \ldots, \alpha_n \in \Gamma$) is $w^{\mathrm{R}} := \alpha_n \ldots \alpha_2 \alpha_1$. The *reversal* of a language $L \subseteq \Gamma^*$ is $L^{\mathrm{R}} := \{w^{\mathrm{R}} \mid w \in L\}$.

Let $v, w \in \Gamma^*$. We say that v is a *subword* of w (denoted by $v \sqsubseteq w$) if we have $w \in v \sqcup\!\sqcup \Gamma^*$. In this case w is a *superword* of v. Note that the induced binary relation $\sqsubseteq$ is a partial ordering on Γ^*. We know that each infix, suffix, and infix is a subword but not vice versa.

The class of regular languages is effectively closed under Boolean operations, product, iteration, homomorphisms and their inverses (including projections), shuffle, reversal, and up- and downclosures wrt. the subword, prefix, suffix, and infix relation. Note that given two NFAs, all of these operations are possible in polynomial time except for complement. However, this operation is efficient[i] if the input is given by a DFA.

2.5 Structures and Logics

A *(relational) signature* is a tuple $\tau := (R, \mathrm{ar})$ where R is a (not necessarily finite) set of *relation names*, and $\mathrm{ar}\colon R \to \mathbb{N}$ is the *arity map*. A *τ-structure* is a tuple $\mathcal{S} = (U, I^{\mathcal{S}})$ where U is a set (the *universe*) and $I^{\mathcal{S}}$ is the *interpretation* of τ, which is a map with $I^{\mathcal{S}}(P) \subseteq U^{\mathrm{ar}(P)}$ for each $P \in R$. In later chapters we denote τ-structures $\mathcal{S}$ as a tuple consisting of its universe U and the images of $I^{\mathcal{S}}$.

Now, let $\tau = (R, \mathrm{ar})$ be a signature and $\mathcal{V}_0$ and $\mathcal{V}_1$ be two countable sets of first-order resp. second-order variables with $\mathcal{V}_0 \cap \mathcal{V}_1 = \varnothing$. For better readability we denote the first-order variables by lowercase letters and the second-order variables by uppercase letters. We define the set of *monadic second-order τ-formulas* (denoted by $\mathsf{MSO}[\tau]$) inductively. Simultaneously, we define the set of *free* variables $\mathrm{fv}(\phi) \subseteq \mathcal{V}_0 \cup \mathcal{V}_1$ of a formula ϕ. First we consider the so-called *atomic formulas*:

- $\top \in \mathsf{MSO}[\tau]$ and $\mathrm{fv}(\top) := \varnothing$,
- $(x = y) \in \mathsf{MSO}[\tau]$ and $\mathrm{fv}(x = y) := \{x, y\}$ where $x, y \in \mathcal{V}_0$,
- $P(x_1, \ldots, x_k) \in \mathsf{MSO}[\tau]$ and $\mathrm{fv}(P(x_1, \ldots, x_k)) := \{x_1, \ldots, x_k\}$ where $P \in R$, $k = \mathrm{ar}(P)$, and $x_1, \ldots, x_k \in \mathcal{V}_0$, and
- $X(x) \in \mathsf{MSO}[\tau]$ and $\mathrm{fv}(X(x)) := \{X, x\}$ where $X \in \mathcal{V}_1$ and $x \in \mathcal{V}_0$.

Now, let $\phi, \psi \in \mathsf{MSO}[\tau]$ be two formulas. Then we also have:

- $\neg\phi \in \mathsf{MSO}[\tau]$ and $\mathrm{fv}(\neg\phi) := \mathrm{fv}(\phi)$,
- $\phi \vee \psi \in \mathsf{MSO}[\tau]$ and $\mathrm{fv}(\phi \vee \psi) := \mathrm{fv}(\phi) \cup \mathrm{fv}(\psi)$,
- $\exists x\colon \phi \in \mathsf{MSO}[\tau]$ and $\mathrm{fv}(\exists x\colon \phi) := \mathrm{fv}(\phi) \setminus \{x\}$ where $x \in \mathcal{V}_0$, and
- $\exists X\colon \phi \in \mathsf{MSO}[\tau]$ and $\mathrm{fv}(\exists X\colon \phi) := \mathrm{fv}(\phi) \setminus \{X\}$ where $X \in \mathcal{V}_1$.

We also write $\phi \wedge \psi$ instead of $\neg(\neg\phi \vee \neg\psi)$, $\forall x\colon \phi$ instead of $\neg\exists x\colon \neg\phi$, and $\forall X\colon \phi$ instead of $\neg\exists X\colon \neg\phi$. A formula ϕ is *first-order* if ϕ contains no occurrence of a second-order variable. The set of all first-order τ-formulas is denoted by $\mathsf{FO}[\tau]$. We also denote the free variables of a formula as its arguments. In other words, for a formula $\phi \in \mathsf{MSO}[\tau]$ with $\mathrm{fv}(\phi) = \{X_1, \ldots, X_m, x_1, \ldots, x_n\}$ we also write $\phi(X_1, \ldots, X_m, x_1, \ldots, x_n)$ instead of ϕ. A formula ϕ is called a *sentence* if $\mathrm{fv}(\phi) = \varnothing$ holds.

[i] In our context, a construction is *efficient* if it requires only polynomial time. In particular, we say that a language is *efficiently regular* if there is an efficient construction of an NFA accepting this language.

The semantics of monadic second-order formulas is defined as follows: let $\mathcal{S} = (U, I^{\mathcal{S}})$ be a τ-structure. An *assignment* of $\mathcal{S}$ is a map $\vartheta\colon \mathcal{V}_0 \cup \mathcal{V}_1 \to U \cup 2^U$ with $\vartheta(x) \in U$ and $\vartheta(X) \subseteq U$ holds for each $x \in \mathcal{V}_0$ and $X \in \mathcal{V}_1$. Then we write for an assignment ϑ and formulas $\phi, \psi \in \mathsf{MSO}[\tau]$:

- $(\mathcal{S}, \vartheta) \vDash \top$
- $(\mathcal{S}, \vartheta) \vDash x = y$ iff $\vartheta(x) = \vartheta(y)$,
- $(\mathcal{S}, \vartheta) \vDash P(x_1, \dots, x_k)$ iff $(\vartheta(x_1), \dots, \vartheta(x_k)) \in I^{\mathcal{S}}(P)$,
- $(\mathcal{S}, \vartheta) \vDash X(x)$ iff $\vartheta(x) \in \vartheta(X)$,
- $(\mathcal{S}, \vartheta) \vDash \neg\phi$ iff $(\mathcal{S}, \vartheta) \nvDash \phi$,
- $(\mathcal{S}, \vartheta) \vDash \phi \vee \psi$ iff $(\mathcal{S}, \vartheta) \vDash \phi$ or $(\mathcal{S}, \vartheta) \vDash \psi$,
- $(\mathcal{S}, \vartheta) \vDash \exists x\colon \phi$ iff there is $s \in U$ with $(\mathcal{S}, \vartheta[x \mapsto s]) \vDash \phi$, and
- $(\mathcal{S}, \vartheta) \vDash \exists X\colon \phi$ iff there is $S \subseteq U$ with $(\mathcal{S}, \vartheta[X \mapsto S]) \vDash \phi$.

For a τ-structure $\mathcal{S}$ and a formula $\phi(\vec{\boldsymbol{X}}, \vec{\boldsymbol{x}}) \in \mathsf{MSO}[\tau]$, we also write $(\mathcal{S}, \vec{\boldsymbol{S}}, \vec{\boldsymbol{s}}) \vDash \phi(\vec{\boldsymbol{X}}, \vec{\boldsymbol{x}})$ if there is an assignment ϑ in $\mathcal{S}$ with $\vartheta(X_i) = S_i$ and $\vartheta(x_i) = s_i$ satisfying $(\mathcal{S}, \vartheta) \vDash \phi$. In particular, if ϕ is a sentence, we also write $\mathcal{S} \vDash \phi$ in this case.

Example 2.5.1. Let $\tau = (\{E\}, \mathrm{ar})$ with $\mathrm{ar}(E) = 2$ be a signature. Then a τ-structure $\mathcal{G}$ (note that this is a directed graph) satisfies

$$\exists x, y, z\colon E(x, y) \wedge E(y, z) \wedge E(z, x)$$

if, and only if, $\mathcal{G}$ contains a cycle of length three. $\mathcal{G}$ satisfies

$$\exists X\colon \big(\exists x\colon X(x) \wedge \exists y\colon \neg X(y) \wedge \forall x, y\colon E(x, y) \to (x \in X \leftrightarrow y \in X)\big)$$

if, and only if, $\mathcal{G}$ is not connected. It is well-known that there is no first-order formula expressing this property (one can easily show this with the help of so-called *Ehrenfeucht-Fraïssé games* [Ehr61, Fra54]). ⌟

Let $\phi \in \mathsf{MSO}[\tau]$ be a formula and $y \in \mathrm{fv}(\phi) \cap \mathcal{V}_0$ be a free first-order variable. Then we write $\phi[y \leftarrow x]$ for the formula which is obtained from ϕ by replacement of any free occurrence of y by x. Now, let $\phi, \xi \in \mathsf{MSO}[\tau]$ be two formulas and $y \in \mathrm{fv}(\xi) \cap \mathcal{V}_0$ be a distinguished free first-order variable of ξ. Then the *restriction $\phi\restriction_\xi$ of ϕ to ξ* is defined by

$$\phi\restriction_\xi := \begin{cases} \phi & \text{if } \phi \text{ is atomic} \\ \neg(\psi\restriction_\xi) & \text{if } \phi = \neg\psi \\ \psi\restriction_\xi \vee \chi\restriction_\xi & \text{if } \phi = \psi \vee \chi \\ \exists x\colon (\psi\restriction_\xi \wedge \xi[y \leftarrow x]) & \text{if } \phi = \exists x\colon \psi \\ \exists X\colon (\psi\restriction_\xi \wedge \forall x\colon X(x) \to \xi[y \leftarrow x]) & \text{if } \phi = \exists X\colon \psi\,. \end{cases}$$

In other words, we restrict the quantifiers in ϕ to the values satisfying ξ. Obviously, we have $\phi|_\xi \in \mathsf{FO}[\tau]$ if $\phi, \xi \in \mathsf{FO}[\tau]$ holds.

Chapter 3

Automata and Storage Mechanisms

In this chapter we want to introduce finite automata having several mechanisms to store data. A possible representation of such mechanism is a *data structure*. A data structure is an object consisting of a (possibly infinite) set of states and a finite number of methods in which we can modify the state of the structure. Famous data structures are linked lists, arrays, strings, or bitmaps. However, our results in the following chapters do not depend on the concrete implementation of such storage mechanism. So, we consider a more abstract model to store data - so-called *(abstract) data types* (cf. [Cle86, Cor+09, SS20]). In our context a data type consists of a (possibly infinite) set of states, a finite signature consisting of functions, so-called *actions*, and the semantics of this signature (i.e., computable functions realizing the actions). Then we can understand an action as the name f of a function $f\colon U \times I \to U \times O$ and their semantics as the concrete function where U is the set of states of our data type, I is a set of inputs, and O is a set of outputs. In other words, depending on the current state of the data type, an action takes an input $i \in I$, modifies its state, and returns an output $o \in O$. Note that in literature, the semantics of an abstract data type is sometimes only implicitly given by a system of *axioms* (cf. for example [SS20]). We will consider such axioms in Chapter 4 for several data types.

Concretely, we focus on so-called transforming data types. The actions of these data types do not have any additional in- and outputs. In other words, the result of the application of such action only depends on the current state of the data type and only yields a new state. We will see later that such data types are suitable for attaching them to finite automata. Then, depending on the current state of the attached data type the automaton is able to do a transition from one configuration to another one and to modify it accordingly. In fact such data type represents an additional memory of the automaton.

First, we formally introduce transforming data types and automata using them as their memory. Later we give some examples and extend the notion to lossy and distributed data types.

3.1 Definitions

We fix the symbol $\perp$ (the so-called *error state*). A *(transforming) data type* is a triple $\mathcal{D} = (U, \Sigma, \Theta)$ where U is a recursive set (the so-called *universe*) with $\perp \notin U$, Σ is an alphabet (the so-called *signature*), and $\Theta\colon \Sigma \times (U \cup \{\perp\}) \to (U \cup \{\perp\})$ is a recursive, total function with $\Theta(\alpha, \perp) = \perp$ for each $\alpha \in \Sigma$ (the *semantics* of the signature). An element $\alpha \in \Sigma$ is also called a *(basic) action*.

Now, let $t \in \Sigma^*$ be a sequence of actions of $\mathcal{D}$. Then we define the map $[\![t]\!]_{\mathcal{D}}\colon (U \cup \{\perp\}) \to (U \cup \{\perp\})$ inductively as follows:

- $[\![\varepsilon]\!]_{\mathcal{D}} := \mathrm{id}_{U\cup\{\perp\}}$ is the identity map on $U \cup \{\perp\}$ and
- if $t = \alpha s$ for $\alpha \in \Sigma$ and $s \in \Sigma^*$ we set $[\![t]\!]_{\mathcal{D}}(x) := [\![s]\!]_{\mathcal{D}}(\Theta(\alpha, x))$ for each $x \in U \cup \{\perp\}$.

In particular, we have $[\![\alpha]\!]_{\mathcal{D}} = \Theta(\alpha, .)$ for each $\alpha \in \Sigma$. For $t \in \Sigma^*$ we call $[\![t]\!]_{\mathcal{D}}$ the *transformation* induced by the action sequence t. We say that "$[\![t]\!]_{\mathcal{D}}(x)$ is undefined" whenever we have $[\![t]\!]_{\mathcal{D}}(x) = \perp$. The set of all transformations $[\![t]\!]_{\mathcal{D}}$ with composition forms a monoid with identity $[\![\varepsilon]\!]_{\mathcal{D}}$. This monoid is called the *transformation monoid* (or *action monoid*) and is denoted by $\mathbb{T}(\mathcal{D})$. Note that we write $[\![t]\!]$ instead of $[\![t]\!]_{\mathcal{D}}$ whenever the situation is clear.

Let $\mathcal{D} = (U, \Sigma, \Theta)$ be a data type. A *$\mathcal{D}$-automaton* is a tuple $\mathfrak{A} = (Q, \Gamma, \mathcal{D}, I, c, \Delta, F)$ where Q is a finite set of *(control) states*, Γ is an alphabet, $I, F \subseteq Q$ are the sets of *initial* and *final states*, respectively, $c \in U$ is the *initial content* and $\Delta \subseteq Q \times (\Gamma \cup \{\varepsilon\}) \times \Sigma^* \times Q$ is a finite set of *transitions*.

Now, let $\mathfrak{A} = (Q, \Gamma, \mathcal{D}, I, c, \Delta, F)$ be a $\mathcal{D}$-automaton. The set of *configurations* of $\mathfrak{A}$ is the set $\mathrm{Conf}_{\mathfrak{A}} := Q \times U$. The *initial and final configurations* of $\mathfrak{A}$ are $\mathrm{Init}_{\mathfrak{A}} := I \times \{c\}$ and $\mathrm{Final}_{\mathfrak{A}} := F \times U$, respectively. The *configuration graph* of $\mathfrak{A}$ is the (possibly infinite) labeled graph $\mathcal{G}_{\mathfrak{A}} = (\mathrm{Conf}_{\mathfrak{A}}, E)$ with $(p, x) \xrightarrow{a}_{\mathcal{G}_{\mathfrak{A}}} (q, y)$ if, and only if, there is a transition $(p, a, t, q) \in \Delta$ with $[\![t]\!](x) = y$ for any two configurations $(p, x), (q, y) \in \mathrm{Conf}_{\mathfrak{A}}$ and for any letter $a \in \Gamma \cup \{\varepsilon\}$. In other words, a transition $(p, a, t, q) \in \Delta$ applies the transition (p, a, q) of the underlying NFA and applies the transformation $[\![t]\!]$ to its attached data type. Note that the application of such transition requires $[\![t]\!](x) \neq \perp$ since $\perp \notin U = Q\,\mathrm{Conf}_{\mathfrak{A}}$. From now on we write $(p, x) \xrightarrow{a}_{\mathfrak{A}} (q, y)$ instead of $(p, x) \xrightarrow{a}_{\mathcal{G}_{\mathfrak{A}}} (q, y)$. The *accepted language* of $\mathfrak{A}$ is

$$L(\mathfrak{A}) := \{w \in \Gamma^* \mid \mathrm{Init}_{\mathfrak{A}} \xrightarrow{w}_{\mathfrak{A}} \mathrm{Final}_{\mathfrak{A}}\}.$$

Remark 3.1.1. Our data types only consist of their universe and some transforming operations. However, there are data types like *counters with zero-tests* or *sets* which contain additional tests. A *test* (or *lookup*) is a function $t\colon U \cup \{\perp\} \to \{0, 1\}$ with $t(\perp) = 0$. Then a $\mathcal{D}$-automaton is allowed to apply a transition labeled with such test t if, and only if, the content $c \in U$ of its data types satisfies $t(c) = 1$.

From the test t we also obtain a (basic) transformation $[\![t']\!]\colon (U \cup \{\perp\}) \to (U \cup \{\perp\})$ with the following semantics:

$$[\![t']\!](c) = \begin{cases} c & \text{if } t(c) = 1 \\ \perp & \text{otherwise}. \end{cases}$$

Then the automaton is able to apply a t'-labeled transition if, and only if, it could apply a t-labeled transition. Hence, our data types are also able to simulate types with tests. ⌟

Remark 3.1.2. Let $\mathcal{D}$ be a data type. The $\mathcal{D}$-automata defined above are not exactly the same as the so-called *valence automata* (see, e.g., [Pău80, FS02]). In such automata we require the universe of $\mathcal{D}$ to be a monoid. Additionally, instead of accepting when reaching a final state, valence automata only accept if their memory is empty (i.e., the content is the monoid's identity). So, for an accepting run from an initial configuration (ι, c) to a final configuration (f, e) we require the existence of a right-inverse element x in the underlying monoid (i.e., we have $cx = e$).

For certain data types, $\mathcal{D}$-automata are as powerful as valence automata with an underlying monoid $\mathbb{M}$. For example, this is possible for stacks or counters. In this case,

we can simulate a $\mathcal{D}$-automaton by a valence automaton having the memory $\mathbb{T}(\mathcal{D})$ and vice versa. So, instead of applying $[\![t]\!]$ in any transition of the $\mathcal{D}$-automaton we compute the composition of the previously applied transformations with the label $[\![t]\!] \in \mathbb{T}(\mathcal{D})$.

However, it is impossible to define, for example, a queue automaton as a valence automaton in this way. This is due to the fact that the transformation monoid of a queue has no non-trivial divisor of its identity (we will see this later in this thesis). ⌟

We are able to model the most important data types with our definitions. Some of them are given in the following section. However, there are data types which cannot be modeled with our definitions. For example, we are unable to define set-theoretical operations (like union, intersection, or difference) of a (*dynamic*) *set* with an infinite universe.

3.2 Examples

3.2.1 Finite Memories

Consider a finite data type $\mathcal{F} = (U, \Sigma, \Theta)$, i.e., the universe U is finite. A typical example for such finite data type is a *bounded queue* (or *bounded channel*). A bounded queue stores sequences of length at most $k \in \mathbb{N}$ of elements from a given alphabet A. In other words, the content of a bounded queue is a word from $A^{\leq k}$. For each letter $a \in A$ a bounded queue has two possible basic actions. On the one hand, we can write the letter a at the end of the bounded queue's content (we denote this action by a). We do this by appending an a to the content. Note that this action fails if we try to write a letter a into a full bounded queue (i.e., if the content already has length k). On the other hand, we can read a from the head position of the queue (we denote this action by $\overline{a}$). In this case, we remove a leading a from the queue's content. This action is undefined if there is no a at the head position of the bounded queue. We assume that the alphabet of read actions $\overline{A} := \{\overline{a} \mid a \in A\}$ is a disjoint copy of the alphabet A.

Formally, a bounded queue is the data type $\mathcal{BQ}_k = (A^{\leq k}, \Sigma, \Theta)$ where $k \in \mathbb{N}$, $\Sigma = A \cup \overline{A}$,

$$[\![a]\!](x) := \begin{cases} xa & \text{if } |x| < k \text{ and } x \neq \bot \\ \bot & \text{otherwise,} \end{cases} \qquad \text{and} \qquad [\![\overline{a}]\!](x) := \begin{cases} y & \text{if } x = ay \\ \bot & \text{otherwise} \end{cases}$$

holds for each $a \in A$ and $x \in A^* \cup \{\bot\}$. Similarly, we may define *bounded stacks* (or *bounded pushdowns*), which only differ in their write action a. In this case, one prepends the letter a to the stack's content instead of appending it.

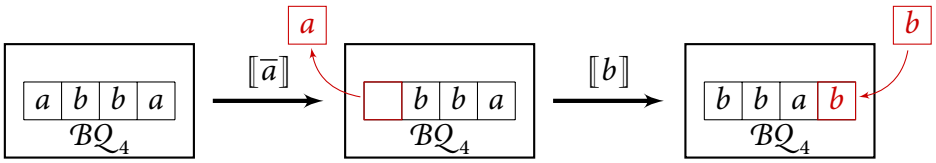

Figure 3.1. A visualization of a (bounded) queue with content *abba* applying the transformation $[\![\overline{a}b]\!]$.

It is well-known that automata having a finite data type as their storage mechanism accept exactly the class of regular languages. To this end, let $\mathfrak{A} = (Q, \Gamma, \mathcal{F}, I, c, \Delta, F)$ be an

$\mathcal{F}$-automaton. Since U is finite, the set $\mathrm{Conf}_{\mathfrak{A}}$ of configurations of $\mathfrak{A}$ is finite as well. Hence, we can construct an NFA with ε-transitions which simulates this finite configuration graph. From $\mathfrak{A}$ we can compute the NFA $\mathfrak{B} = (Q', \Gamma, I', \Delta', F')$ with:

- $Q' := \mathrm{Conf}_{\mathfrak{A}} = Q \times U$,
- $I' := \mathrm{Init}_{\mathfrak{A}} = I \times \{c\}$,
- $\Delta' := \{((p,x), a, (q,y)) \mid \exists (p,a,t,q) \in \Delta : [\![t]\!](x) = y\}$, and
- $F' := \mathrm{Final}_{\mathfrak{A}} = F \times U$.

It is a simple exercise to prove that $L(\mathfrak{A}) = L(\mathfrak{B})$ holds.

Conversely, from any given NFA $\mathfrak{B}$ we can construct an $\mathcal{F}$-automaton $\mathfrak{A}$ accepting $L(\mathfrak{B})$. A possible idea is to construct $\mathfrak{A}$ such that it does not modify the content of its memory. In other words, each transition is labeled with the transformation $[\![\varepsilon]\!] = \mathrm{id}_{U \cup \{\bot\}}$.

All in all, we see that for any finite data type $\mathcal{F}$ the $\mathcal{F}$-automata accept exactly the class of regular languages. In particular, automata with a bounded queue or a bounded stack are as powerful as NFAs (without additional memory).

3.2.2 Counters

A *(partially blind) counter* is a data type which has a natural number as its content and is allowed to increase or decrease this number. Since the content is non-negative, the decrease action on the content 0 is undefined. Formally, a counter is the data type $C_1 = (\mathbb{N}, \{i, d\}, \Theta)$ where

$$[\![i]\!](x) := \begin{cases} x+1 & \text{if } x \in \mathbb{N} \\ \bot & \text{otherwise} \end{cases} \quad \text{and} \quad [\![d]\!](x) := \begin{cases} x-1 & \text{if } x \in \mathbb{N} \setminus \{0\} \\ \bot & \text{otherwise} \end{cases}$$

for each $x \in \mathbb{N} \cup \{\bot\}$. In other words, i increments the counter and d decrements the counter.

It is easy to see that $[\![id]\!](x) = x = \mathrm{id}_{\mathbb{N}\cup\{\bot\}}(x)$ holds for each $x \in \mathbb{N} \cup \{\bot\}$ and $[\![di]\!](0) = \bot \neq 0 = \mathrm{id}_{\mathbb{N}\cup\{\bot\}}(0)$. Hence, $[\![d]\!]$ is the right-inverse of $[\![i]\!]$ in the transformation monoid of C_1 but it is not the left-inverse of $[\![i]\!]$ implying that the transformation monoid $\mathbb{T}(C_1)$ is not a group. By [Loto2] this monoid is exactly the so-called *bicyclic semigroup* which was first introduced by Lyapin [Lya53]. The C_1-automata are exactly the *(partially blind) one-counter automata* [Gre78].

For a positive integer $k > 0$, we can also model k (partially blind) counters, which consist of a k-tuple of natural numbers and each number can be increased or decreased. This can be formalized by the data type $C_k = (\mathbb{N}^k, \{i_j, d_j \mid 1 \leq j \leq k\}, \Theta)$ where $[\![i_j]\!]$ increases the counter j and $[\![d_j]\!]$ decreases this counter. In this case, C_k-automata are the *(partially blind) k-counter automata* (in literature these are also known as *vector addition systems* [KM69] or *Petri nets* [Pet62]).

Similarly, we can model *k blind counters*. The contents of this data type are k-tuples of integers, i.e., their contents may also be negative. This can be formalized by the data type $\mathcal{Z}_k = (\mathbb{Z}^k, \{i_j, d_j \mid 1 \leq j \leq k\}, \Theta)$. In this case, we have $[\![i_j d_j]\!] = \mathrm{id}_{\mathbb{Z}^k \cup \{\bot\}} = [\![d_j i_j]\!]$. Hence, the transformation monoid $\mathbb{T}(\mathcal{Z}_k)$ is isomorphic to the group $\mathbb{Z}^k$. The $\mathcal{Z}_k$-automata are exactly the *blind k-counter automata*.

Counters with zero-tests are k partially blind counters which additionally are allowed to check their contents for zero. This can be modeled by the data type $\mathcal{D}_k = (\mathbb{N}^k, \{i_j, d_j, z_j \mid$

$1 \leq j \leq k\}, \Theta)$ where $[\![z_j]\!](x_1, \ldots, x_k) = (x_1, \ldots, x_k)$ holds if $x_j = 0$ and $[\![z_j]\!](x_1, \ldots, x_k) = \bot$ holds otherwise. Note that the only difference to partially blind counters is the addition of the tests z_j. However, for $k \geq 2$, $\mathcal{D}_k$-automata (so-called *Minsky-machines*) are much more powerful than $\mathcal{C}_k$-automata. Concretely, $\mathcal{D}_k$-automata accept exactly the recursively enumerable languages [Min67].

3.2.3 Stacks

A very important basic data type is an unbounded *(lifo-)stack* (or *pushdown*). A stack stores sequences of elements from a given alphabet A. In contrast to bounded queues or stacks, these sequences can have arbitrary finite length. In other words, a content of a stack is a word from A^*. For each letter $a \in A$ we have two possible basic actions: writing the letter a on top of the stack (denoted by a) and reading the letter a from the stack's top position (denoted by $\overline{a}$). In contrast to the bounded case, the write action is defined in any case. However, read actions are still undefined if there is no a on top of the stack's content.

In practice, a computer needs a stack to execute a program with nested function calls. The computer stores its context in this stack when calling a function. Also there are calculators employing the reverse Polish notation (for example one writes $1\,2+$ instead of $1+2$) with the help of a stack in which one stores operands and intermediate values.

Formally, a *stack* is the data type $\mathcal{P}_A = (A^*, \Sigma, \Theta)$ where A is an alphabet, $\Sigma = A \cup \overline{A}$,

$$[\![a]\!](x) := \begin{cases} ax & \text{if } x \in A^* \\ \bot & \text{otherwise}, \end{cases} \qquad \text{and} \qquad [\![\overline{a}]\!](x) := \begin{cases} y & \text{if } x = ay \\ \bot & \text{otherwise} \end{cases}$$

holds for each $a \in A$ and $x \in A^* \cup \{\bot\}$.

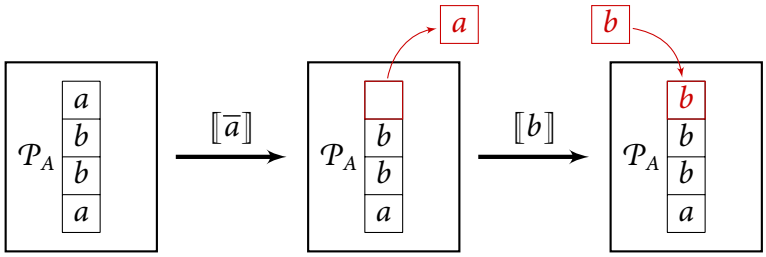

Figure 3.2. A visualization of a stack with content $abba$ applying the transformation $[\![\overline{a}b]\!]$.

Remark 3.2.1. Our definition of a stack slightly differs from the definition of stacks in literature. For example, in [SS20] a stack has only one read action which takes a stack's content and returns the top entry of the stack and the stack's content without this top entry. However, this data type is not transforming. In contrast, the data type as defined above has a read action $\overline{a}$ for each entry $a \in A$. Its corresponding the transformation $[\![\overline{a}]\!]$ verifies that the top entry is actually a and returns afterwards the stack's content without this top entry. We use this variation due to technical reasons. ⌟

We assume that the alphabet of read actions $\overline{A} := \{\overline{a} \mid a \in A\}$ is a disjoint copy of the alphabet A. For a word $t = a_1 a_2 \ldots a_n$ with $a_1, a_2, \ldots, a_n \in A$ we write $\overline{t} := \overline{a_1}\,\overline{a_2} \ldots \overline{a_n}$. Additionally, for $L \subseteq A^*$ we write $\overline{L} := \{\overline{t} \mid t \in L\}$.

Read and write actions of a stack are in some sense generalizations of the increase and decrease actions of a single counter. So, if the underlying alphabet A of stack contents is a singleton, the stack $\mathcal{P}_A$ is essentially a single partially blind counter. Moreover, we have $\mathbb{T}(\mathcal{P}_A) \cong \mathbb{T}(C_1)$ in this case. If A is not a singleton, $\mathbb{T}(C_1)$ is isomorphic to a proper submonoid of $\mathbb{T}(\mathcal{P}_A)$. However, similar to situation in $\mathbb{T}(C_1)$ the read transformation $[\![\overline{a}]\!]$ is the right-inverse element of the corresponding write transformation $[\![a]\!]$ (but not vice versa): for $a \in A$ we have $[\![a\overline{a}]\!] = \mathrm{id}_{A^* \cup \{\bot\}}$ and $[\![\overline{a}a]\!](\varepsilon) = \bot \neq \varepsilon = \mathrm{id}_{A^* \cup \{\bot\}}(\varepsilon)$. Due to this fact, in literature the transformation monoid $\mathbb{T}(\mathcal{P}_A)$ of a stack is also called the *polycyclic semigroup* [NP70].

Note that we can also simulate a single counter with zero-tests $\mathcal{D}_1$ with the help of a stack $\mathcal{P}_A$ with a binary alphabet $A = \{a, \#\}$. In this case, the letter # marks the bottom of the stack. While increasing and decreasing of the counter correspond to writing and reading the letter a, we can simulate a zero-test with the sequence $\overline{\#}\#$.

In general, the $\mathcal{P}_A$-automata are essentially *pushdown automata* (PDA, for short). In literature (like in [HMU01]) each transition of a PDA applies a transformation $[\![t]\!]$ with $t \in \overline{A}A^*$ to their stack. However, we are able (by splitting edges) to construct from a given $\mathcal{P}_A$-automaton an equivalent classical PDA. Hence, our $\mathcal{P}_A$-automata accept exactly the context-free languages and from now on we call them PDA, too.

It is also possible to simulate stacks storing entries from a countable universe U. To this end, we use a uniquely decipherable function $c\colon U^* \to \{0,1\}^*$ (i.e., c is injective). Then writing an $x \in U$ corresponds to $[\![c(x)^{\mathrm{R}}]\!]$ in $\mathcal{P}_{\{0,1\}}$ and reading $x \in U$ corresponds to $[\![\overline{c(x)}]\!]$.

We may also extend this data type to lossy or distributed (multi)-stacks. We will see the definitions of these variations at a later point in this chapter.

3.2.4 Queues

The definition of an unbounded *(fifo-)queue* (or *channel*) is very similar to the ones of a bounded queue and of an unbounded stack. So, the content of a queue is a finite sequence of elements from an alphabet A. We can write and read an element a. However, we do not prepend the a to the queue's content. Instead we append this letter. Anyway, the write action is defined for each queue content.

In practice, a queue can be used to model a single communication channel between two network instances. Also operating systems sometimes use a queue to buffer data for later uses.

Formally, a *queue* is defined as the following data type: $\mathcal{Q}_A = (A^*, \Sigma, \Theta)$ where A is an alphabet, $\Sigma = A \cup \overline{A}$,

$$[\![a]\!](x) := \begin{cases} xa & \text{if } x \in A^* \\ \bot & \text{otherwise}, \end{cases} \quad \text{and} \quad [\![\overline{a}]\!](x) := \begin{cases} y & \text{if } x = ay \\ \bot & \text{otherwise} \end{cases}$$

holds for each $a \in A$ and $x \in A^* \cup \{\bot\}$. Note that we have $[\![a]\!](x) = xa$ and $[\![\overline{a}]\!](ax) = x$. By induction on the length of an action sequence $t \in \Sigma^*$ we observe the following:

Observation 3.2.2. *Let A be an alphabet, $t \in \Sigma^*$ be an action sequence, and $x, y \in A^*$ with $[\![t]\!](x) = y \neq \bot$. Then we have $xw = ry$ where $w = \pi_A(t)$ and $\overline{r} = \pi_{\overline{A}}(t)$.* ◂

The transformation monoid $\mathbb{T}(\mathcal{Q}_A)$ is the so-called *queue monoid* which was first introduced by Huschenbett et al. in [HKZ17].

Note that if A is a singleton, the queue $\mathcal{Q}_A$ degenerates to a partially blind counter. Hence, we have $\mathcal{Q}_A \cong C_1$ in this case. Now, let A be at least binary. Recall that the read transformation of the letter a is the right-inverse element of the corresponding write transformation in the transformation monoid $\mathbb{T}(\mathcal{P}_A)$ of a stack with entries from A. This fact does not hold in the queue monoid. For example, if $a, b \in A$ are two distinct letters we have

$$[\![a\overline{a}]\!](ab) = [\![\overline{a}]\!](aba) = ba \neq ab = \mathrm{id}_{A^* \cup \{\bot\}}(ab)\,,$$

i.e., we have $[\![a\overline{a}]\!] \neq \mathrm{id}_{A^* \cup \{\bot\}}$. In fact, no transformation $[\![t]\!] \in \mathbb{T}(\mathcal{Q}_A) \setminus \{[\![\varepsilon]\!]\}$ has a left- or right-inverse element in this monoid (cf. [HKZ17, Theorem 5.3]).

It is well-known that $\mathcal{Q}_A$-automata (so-called *queue automata*) with an at least binary alphabet A of queue entries are much more powerful than pushdown automata. Concretely, queue automata are able to simulate Turing-machines (cf. [BZ83]). Hence, these automata accept exactly the class of recursively enumerable languages.

Note that again we are able to simulate queues storing entries from a countable universe U. Later we will see the definitions of lossy queues and distributed multi-queues.

3.2.5 Priority Queues

Now, we want to introduce priority queues. These are queues in which we annotate each queue entry with a priority. Entries having a high priority are able to supersede or overtake entries with lower priority. In other words, we are able to bypass the fifo-principle of the queues with the help of these priorities. Similar to stacks and queues, a priority queue is a very important basic data type. It is used by network routers for management of a limited bandwidth or by operating systems for scheduling tasks.

Here, we model the priorities of the queue entries with the help of a quasi-ordering. To this end, let A be an alphabet of queue entries and $\leq$ be a quasi-ordering on A. Then "$a \leq b$" means that b has a higher (or equal) priority than a. Then a *priority queue with internal superseding semantics* is the data type $\mathcal{PQ}^I_{A,\leq} = (A^*, \Sigma, \Theta)$ where

$$[\![a]\!](x) := \begin{cases} xa & \text{if } x \in A^* \\ \bot & \text{otherwise} \end{cases} \text{ and } [\![\overline{a}]\!](x) := \begin{cases} z & \text{if } x = yaz,\ y \in \{b \in A \mid b \leq a, b \neq a\}^* \\ \bot & \text{otherwise} \end{cases}$$

holds for each $a \in A$ and $x \in A^* \cup \{\bot\}$. In other words, these queues are able to remove letters with low priority to allow reading of letters having a higher priority.

We can also define *priority queues with strict superseding semantics* $\mathcal{PQ}^S_{A,\leq}$ which only differ in their read actions:

$$[\![\overline{a}]\!](x) := \begin{cases} z & \text{if } x = yaz,\ y \in \{b \in A \mid b < a\}^* \\ \bot & \text{otherwise}\,. \end{cases}$$

Here, we are able to remove some letters of low priority to read a letter of strictly higher priority.

A *priority queue with overtaking semantics* is the data type $\mathcal{PQ}^O_{A,\leq}$ with the following read actions:

$$[\![\overline{a}]\!](x) := \begin{cases} yz & \text{if } x = yaz,\ y \in \{b \in A \mid b < a\}^* \\ \bot & \text{otherwise}\,. \end{cases}$$

In this case, we are able to read letters a with strictly higher priority but we do not remove the letters with lower priority in front of a.

Haase et al. have shown in [HSS14] that $\mathcal{PQ}^{S}_{A,\leq}$- and $\mathcal{PQ}^{O}_{A,\leq}$-automata are much more powerful than $\mathcal{PQ}^{I}_{A,\leq}$-automata. Concretely, $\mathcal{PQ}^{S}_{A,\leq}$- and $\mathcal{PQ}^{O}_{A,\leq}$-automata are Turing-complete and, hence, accept exactly the recursively enumerable languages.

3.2.6 Turing-Tapes

We can also model Turing-machines with the help of a data type. A *Turing-tape* is a data type consisting of an infinite sequence of letters and a position of the machine's head. We are able to read or replace the letter at the head's position. Additionally, we can move the head by one position to the left or right. Since the machine's head reaches only a finite number of positions of its tape after a finite computation, we assume that almost all positions on its tape are empty (represented by a *blank symbol* $\square$).

We can model such *Turing-tape* by a data type of the form $\mathcal{T}_{A,\square} = (U, \Sigma, \Theta)$ where A is an at least binary alphabet, $\square \in A$ is the *blank symbol*,

$$U = \{f\colon \mathbb{Z} \to A \mid \exists n_- < 0 < n_+ \forall i \in \mathbb{Z} \setminus [n_-, n_+]\colon f(i) = \square\} \times \mathbb{Z},$$

$\Sigma = \{\overline{a}, a \mid a \in A\} \cup \{L, N, R\}$ with

$$[\![a]\!]((f,n)) := (f[n \mapsto a], n), \qquad [\![\overline{a}]\!]((f,n)) := \begin{cases} (f,n) & \text{if } f(n) = a \\ \bot & \text{otherwise,} \end{cases}$$

$[\![L]\!]((f,n)) := (f, n-1)$, $[\![N]\!]((f,n)) := (f,n)$, and $[\![R]\!]((f,n)) := (f, n+1)$ for any $(f,n) \in U$.

We call a $\mathcal{T}_{A,\square}$-automaton a *Turing-machine*. Note that in literature any edge of a classical Turing-machine is labeled by an action sequence from $\overline{A}A\{L,N,R\}$. However, from a given $\mathcal{T}_{A,\square}$-automaton we obtain an equivalent classical Turing-machine by splitting the more general transitions.

For a configuration $(q,(f,n))$ of a Turing-machine we can also write vqw where vw is a subsequence of the image of f containing all non-blank symbols such that the head is on the position of w's first letter.

It is well-known that Turing-machines accept exactly the class of recursively enumerable languages.

3.3 Lossy Data Types

Until now, we have only considered memories which are reliable. Now, we want to consider automata with storage mechanisms which are able to forget any part of their contents at any time. We model the loss of data with the help of a quasi ordering $\leq$ on the data type's universe. Whenever we perform an action of the data type, it is allowed to nondeterministically decrease (wrt. $\leq$) its content before and after the application.

Formally, a *lossy data type* is a tuple $\mathcal{D} = (U, \Sigma, \Theta, \leq)$ where (U, Σ, Θ) is a data type and $\leq$ is a recursive quasi ordering on $U \cup \{\bot\}$ with $\leq \subseteq U^2 \cup \{(\bot, \bot)\}$.

Let $t \in \Sigma$ be an action sequence. We define the *transformation* $[\![t]\!]_{\mathcal{D}}\colon (U \cup \{\bot\}) \to 2^{U \cup \{\bot\}}$ induced by t inductively as follows:

- $[\![\varepsilon]\!]_{\mathcal{D}}(x) := \downarrow_{\leq} x$,
- $[\![\alpha]\!]_{\mathcal{D}}(x) := \{y \in U \cup \{\bot\} \mid \exists x' \leq x \colon y \leq \Theta(\alpha, x')\}$ for $\alpha \in \Sigma$, and
- if $t = \alpha s$ for $\alpha \in \Sigma$ and $s \in \Sigma^*$ we set

$$[\![t]\!]_{\mathcal{D}}(x) := \bigcup_{y \in [\![\alpha]\!]_{\mathcal{D}}(x)} [\![s]\!]_{\mathcal{D}}(y) = [\![s]\!]_{\mathcal{D}}([\![\alpha]\!]_{\mathcal{D}}(x))$$

for each $x \in U \cup \{\bot\}$. In other words, the transformation $[\![\alpha_1 \dots \alpha_k]\!]_{\mathcal{D}}$ is the application of the basic actions $\alpha_1, \dots, \alpha_k \in \Sigma$ with some intermediate loss of data. From now on we omit the index of $[\![t]\!]_{\mathcal{D}}$ and write $[\![t]\!]$ whenever the situation is clear.

For two sequences $s, t \in \Sigma^*$ we can define the following operation:

$$([\![s]\!] \circ [\![t]\!])(x) := \bigcup_{y \in [\![s]\!](x)} [\![t]\!](y)$$

for each $x \in U \cup \{\bot\}$. Then, we observe the following properties of $\circ$:

Observation 3.3.1. *Let $\mathcal{D} = (U, \Sigma, \Theta, \leq)$ be a lossy data type. Then the following statements hold:*

(1) for each $s, t \in \Sigma^*$ we have $[\![st]\!] = [\![s]\!] \circ [\![t]\!]$.

(2) for each $t \in \Sigma^*$ we have $[\![t]\!] = [\![\varepsilon]\!] \circ [\![t]\!] = [\![t]\!] \circ [\![\varepsilon]\!]$. ◂

Hence, the set of all transformations $[\![t]\!]$ with $\circ$ forms a monoid with identity $[\![\varepsilon]\!]$. We call this monoid the *transformation monoid* $\mathbb{T}(\mathcal{D})$ of $\mathcal{D}$.

Finally, *lossy $\mathcal{D}$-automata* $\mathfrak{A}$ are defined similar to the non-lossy automata. The only difference can be found in the definition of the configuration graph. So, for two configurations $(p, x), (q, y) \in \mathrm{Conf}_{\mathfrak{A}}$ and $a \in \Gamma \cup \{\varepsilon\}$ we write $(p, x) \xrightarrow{a}_{\mathfrak{A}} (q, y)$ if there exists a transition $(p, a, t, q) \in \Delta$ with $y \in [\![t]\!](x)$. Additionally, we write $(p, x) \xrightarrow{\varepsilon}_{\mathfrak{A}} (q, y)$ if $p = q$ and $y \in [\![\varepsilon]\!](x)$ (i.e., $y \leq x$) holds. Hence, the configuration graph $\mathcal{G}_{\mathfrak{A}}$ contains more edges than the configuration graph of a non-lossy automaton.

3.3.1 Partially Lossy Stacks

Partially lossy stacks are stacks which are allowed to forget some parts of their contents at any time. With the help of such partially lossy stack we are able to model programs with nested function calls which are allowed to raise and handle errors (in many programming languages these errors are called *exceptions*). So, whenever a function raises such error, the computer aborts its program flow and jumps to another function in the call stack which is able to handle this error. In this case, the function call stack loses any function between the one raising the error and the one handling this error. Hence, we can understand the functions which are unable to handle errors as forgettable entries in the stack's content and the remaining ones as unforgettable entries.

This partial lossiness is specified with the help of a so-called lossiness alphabet which is defined as follows:

Definition 3.3.2. A *lossiness alphabet* is a tuple $\mathcal{L} = (F, U)$ where $F \cap U = \varnothing$ and $A_{\mathcal{L}} := F \cup U$ is an alphabet. We also write A instead of $A_{\mathcal{L}}$. Here, we say that F is the set of *forgettable* letters and U is the set of *unforgettable* letters.

Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $x, y \in A^*$ be contents. We say that x is an $\mathcal{L}$-*subword* of y (denoted by $x \sqsubseteq_{\mathcal{L}} y$) if $\pi_U(y) \sqsubseteq x \sqsubseteq y$ holds (where $\sqsubseteq$ is the subword ordering on A). ⌟

Note that $\sqsubseteq_{\mathcal{L}}$ is a partial ordering for any lossiness alphabet $\mathcal{L}$. In particular, $\sqsubseteq_{(\varnothing,U)}$ is the equality relation on A^* and $\sqsubseteq_{(F,\varnothing)}$ is the subword relation $\sqsubseteq$ on A^*.

Let $\mathcal{L} = (F, U)$ be a lossiness alphabet. A *partially lossy stack* (or *pls*, for short) is the lossy data type $\mathcal{PLS}_{\mathcal{L}} = (A^*, \Sigma, \Theta, \sqsubseteq_{\mathcal{L}})$ where (A^*, Σ, Θ) is the data type $\mathcal{P}_A$ of a stack. In other words, the only difference between a stack $\mathcal{P}_A$ and a partially lossy stack $\mathcal{PLS}_{\mathcal{L}}$ is the definition of $[\![.]\!]$.

If $F = \varnothing$, we see $[\![t]\!]_{\mathcal{PLS}_{(\varnothing,U)}}(x) = \{[\![t]\!]_{\mathcal{P}_A}(x)\}$ for any action sequence $t \in \Sigma^*$ and any stack content $x \in A^* \cup \{\bot\}$. Hence, $\mathcal{PLS}_{(\varnothing,U)}$ is fully reliable and coincides with a stack $\mathcal{P}_A$ as defined in Section 3.2.3. In contrast, if $U = \varnothing$ the structure $\mathcal{PLS}_{(F,\varnothing)}$ models a fully lossy stack. Anyway the transformation monoid $\mathbb{T}(\mathcal{PLS}_{\mathcal{L}})$ is called the *partially lossy stack monoid* (or *pls monoid*, for short). Accordingly, $\mathcal{PLS}_{\mathcal{L}}$-automata are called *partially lossy stack automata* (or *pls automata*).

Next, we will see that, for any lossiness alphabet $\mathcal{L} = (F, U)$, there is a (non-lossy) data type having a transformation monoid isomorphic to $\mathbb{T}(\mathcal{PLS}_{\mathcal{L}})$. So, consider the data type $\mathcal{P}_{\mathcal{L}} := (A^*, \Sigma, \Theta)$ where $\Sigma = A \cup \overline{A}$,

$$[\![a]\!]_{\mathcal{P}_{\mathcal{L}}}(x) := \begin{cases} ax & \text{if } x \in A^* \\ \bot & \text{otherwise}, \end{cases} \quad \text{and} \quad [\![\overline{a}]\!]_{\mathcal{P}_{\mathcal{L}}}(x) := \begin{cases} z & \text{if } x = yaz, y \in (F \setminus \{a\})^* \\ \bot & \text{otherwise} \end{cases}$$

for each $a \in A$ and $x \in A^* \cup \{\bot\}$. In other words, whenever we read a letter a from the content we are allowed to forget some other letters from F which are in front of a. This means, such stack forgets only those letters blocking the read action of another letter. We can call this behavior the "*read-lossy*" semantics of a partially lossy stack[ii]. We will show next that this semantics is equivalent to the "default" semantics described by $\mathcal{PLS}_{\mathcal{L}}$. Concretely, we will prove $\mathbb{T}(\mathcal{PLS}_{\mathcal{L}}) \cong \mathbb{T}(\mathcal{P}_{\mathcal{L}})$. To this end, we first prove that $[\![t]\!]_{\mathcal{P}_{\mathcal{L}}}$ is monotonic with respect to $\sqsubseteq_{\mathcal{L}}$:

Lemma 3.3.3. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $t \in \Sigma^*$ be an action sequence, and $x, y \in A^*$ be two words with $x \sqsubseteq_{\mathcal{L}} y$ and $[\![t]\!]_{\mathcal{P}_{\mathcal{L}}}(x) \neq \bot$. Then $[\![t]\!]_{\mathcal{P}_{\mathcal{L}}}(x) \sqsubseteq_{\mathcal{L}} [\![t]\!]_{\mathcal{P}_{\mathcal{L}}}(y) \neq \bot$ holds.*

Proof. We prove the claim by induction on the length of the action sequence $t \in \Sigma^*$. If $t = \varepsilon$, then we have

$$[\![\varepsilon]\!]_{\mathcal{P}_{\mathcal{L}}}(x) = x \sqsubseteq_{\mathcal{L}} y = [\![\varepsilon]\!]_{\mathcal{P}_{\mathcal{L}}}(y).$$

[ii] This definition is according to the "read-lossy" semantics of a partially lossy queue as described in [KKP18] (note that we will recall this lossy data type in the next subsection). In a similar way, Chambart and Schnoebelen presented the "write-lossy" semantics of a lossy queue in [CS08, Appendix A]. In this semantics the lossy queue omits write actions whenever the corresponding letters will get lost in the succeeding computation.

Let $t = a \in A$. Then we have $[\![a]\!]_{\mathcal{P}_\mathcal{L}}(x) = ax \sqsubseteq_\mathcal{L} ay = [\![a]\!]_{\mathcal{P}_\mathcal{L}}(y)$.

Now, let $t = \overline{a} \in \overline{A}$. By $[\![\overline{a}]\!]_{\mathcal{P}_\mathcal{L}}(x) \neq \bot$ there is $x' \in (F \setminus \{a\})^*$ with $x = x'a \cdot [\![\overline{a}]\!]_{\mathcal{P}_\mathcal{L}}(x)$. Due to $x \sqsubseteq_\mathcal{L} y$ there are words $y', y'' \in A^*$ with $y = y'ay''$, $x' \sqsubseteq_\mathcal{L} y'$, and $[\![\overline{a}]\!]_{\mathcal{P}_\mathcal{L}}(x) \sqsubseteq_\mathcal{L} y''$. Then we have to consider two cases:

(1) $y' \in (F \setminus \{a\})^*$ holds. Then we have $[\![\overline{a}]\!]_{\mathcal{P}_\mathcal{L}}(y'ay'') = y''$ and, therefore, we learn

$$[\![\overline{a}]\!]_{\mathcal{P}_\mathcal{L}}(x) \sqsubseteq_\mathcal{L} y'' = [\![\overline{a}]\!]_{\mathcal{P}_\mathcal{L}}(y)\,.$$

(2) $y' \notin (F \setminus \{a\})^*$ holds. From $x' \sqsubseteq_\mathcal{L} y'$ and $x' \in F^*$ we infer $y' \in F^*$. Note that in this case $y' \notin (F \setminus \{a\})^*$ implies $a \in F$. There are words $z \in (F \setminus \{a\})^*$ and $z' \in F^*$ such that $y' = zaz'$, i.e., we have $y = zaz'ay''$. Then we see $[\![\overline{a}]\!]_{\mathcal{P}_\mathcal{L}}(y) = z'ay''$ implying

$$[\![\overline{a}]\!]_{\mathcal{P}_\mathcal{L}}(x) \sqsubseteq_\mathcal{L} y'' \sqsubseteq_\mathcal{L} z'ay'' = [\![\overline{a}]\!]_{\mathcal{P}_\mathcal{L}}(y)\,.$$

Finally, let $t = \alpha s$ for a basic action $\alpha \in \Sigma$ and an action sequence $s \in \Sigma^+$. Then by the induction hypothesis we obtain $[\![\alpha]\!]_{\mathcal{P}_\mathcal{L}}(x) \sqsubseteq_\mathcal{L} [\![\alpha]\!]_{\mathcal{P}_\mathcal{L}}(y)$. This implies the following:

$$\begin{aligned} [\![t]\!]_{\mathcal{P}_\mathcal{L}}(x) &= [\![s]\!]_{\mathcal{P}_\mathcal{L}}([\![\alpha]\!]_{\mathcal{P}_\mathcal{L}}(x)) \\ &\sqsubseteq_\mathcal{L} [\![s]\!]_{\mathcal{P}_\mathcal{L}}([\![\alpha]\!]_{\mathcal{P}_\mathcal{L}}(y)) && \text{(by induction hypothesis)} \\ &= [\![t]\!]_{\mathcal{P}_\mathcal{L}}(y)\,. \end{aligned}$$

◂

Let $x \in A^*$ be a word and $t \in \Sigma^*$ be an action sequence with $[\![t]\!]_{\mathcal{P}_\mathcal{L}}(x) \neq \bot$. By definition of $\mathcal{P}_\mathcal{L}$ we can see that $[\![t]\!]_{\mathcal{P}_\mathcal{L}}(x) \in [\![t]\!]_{\mathcal{PLS}_\mathcal{L}}(x)$ holds. In other words, the application of t to x in $\mathcal{P}_\mathcal{L}$ is a possible way to apply t to x in a partially lossy stack $\mathcal{PLS}_\mathcal{L}$. We will show next, that this word is the maximal stack content (wrt. $\sqsubseteq_\mathcal{L}$) which is a possible result of $\mathcal{PLS}_\mathcal{L}$ on application of t to x.

Proposition 3.3.4. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $t \in \Sigma^*$ be an action sequence, and $x \in A^*$ be a word. Then we have*

$$[\![t]\!]_{\mathcal{PLS}_\mathcal{L}}(x) = {\downarrow_{\sqsubseteq_\mathcal{L}}}([\![t]\!]_{\mathcal{P}_\mathcal{L}}(x))\,.$$

In particular, for any $s, t \in \Sigma^*$ and $x \in A^*$ we have

$$[\![s]\!]_{\mathcal{PLS}_\mathcal{L}}(x) = [\![t]\!]_{\mathcal{PLS}_\mathcal{L}}(x) \iff [\![s]\!]_{\mathcal{P}_\mathcal{L}}(x) = [\![t]\!]_{\mathcal{P}_\mathcal{L}}(x)\,.$$

Proof. We prove that

$$y \in [\![t]\!]_{\mathcal{PLS}_\mathcal{L}}(x) \iff y \sqsubseteq_\mathcal{L} [\![t]\!]_{\mathcal{P}_\mathcal{L}}(x)$$

holds for each $x, y \in A^*$ and $t \in \Sigma^*$.

First, we show the implication "$\Rightarrow$" by induction on the length of the word t. The case $t = \varepsilon$ is obvious since $y \in [\![\varepsilon]\!]_{\mathcal{PLS}_\mathcal{L}}(x)$ implies $y \sqsubseteq_\mathcal{L} x$. Then we infer $y \sqsubseteq_\mathcal{L} x = [\![\varepsilon]\!]_{\mathcal{P}_\mathcal{L}}(x)$.

Now, let $|t| \geq 1$, i.e., there are $\alpha \in \Sigma$ and $s \in \Sigma^*$ with $t = \alpha s$. Let $y \in [\![t]\!]_{\mathcal{PLS}_\mathcal{L}}(x)$. Then there is $z \in [\![\alpha]\!]_{\mathcal{PLS}_\mathcal{L}}(x)$ with $y \in [\![s]\!]_{\mathcal{PLS}_\mathcal{L}}(z)$. From $z \in [\![\alpha]\!]_{\mathcal{PLS}_\mathcal{L}}(x)$ we infer the existence of $x', z' \in A^*$ with $x' \sqsubseteq_\mathcal{L} x$, $\Theta(\alpha, x') = z'$, and $z \sqsubseteq_\mathcal{L} z'$. From $y \in [\![s]\!]_{\mathcal{PLS}_\mathcal{L}}(z)$ we infer by induction hypothesis $y \sqsubseteq_\mathcal{L} [\![s]\!]_{\mathcal{P}_\mathcal{L}}(z)$. Next we consider the following two cases:

(1) $\alpha = a \in A$. Then from $\Theta(a, x') = z'$ we obtain $z' = ax'$ and therefore

$$\begin{aligned} y &\sqsubseteq_{\mathcal{L}} [\![s]\!]_{\mathcal{P}_{\mathcal{L}}}(z) \sqsubseteq_{\mathcal{L}} [\![s]\!]_{\mathcal{P}_{\mathcal{L}}}(z') && \text{(by Lemma 3.3.3)} \\ &= [\![s]\!]_{\mathcal{P}_{\mathcal{L}}}(ax') = [\![as]\!]_{\mathcal{P}_{\mathcal{L}}}(x') \\ &\sqsubseteq_{\mathcal{L}} [\![t]\!]_{\mathcal{P}_{\mathcal{L}}}(x)\,. && \text{(by Lemma 3.3.3)} \end{aligned}$$

(2) $\alpha = \overline{a} \in \overline{A}$. Then from $\Theta(\overline{a}, x') = z'$ we obtain $x' = az'$ and therefore

$$\begin{aligned} y &\sqsubseteq_{\mathcal{L}} [\![s]\!]_{\mathcal{P}_{\mathcal{L}}}(z) \sqsubseteq_{\mathcal{L}} [\![s]\!]_{\mathcal{P}_{\mathcal{L}}}(z') && \text{(by Lemma 3.3.3)} \\ &= [\![\overline{a}s]\!]_{\mathcal{P}_{\mathcal{L}}}(az') = [\![t]\!]_{\mathcal{P}_{\mathcal{L}}}(x') \\ &\sqsubseteq_{\mathcal{L}} [\![t]\!]_{\mathcal{P}_{\mathcal{L}}}(x)\,. && \text{(by Lemma 3.3.3)} \end{aligned}$$

Towards the converse implication "$\Leftarrow$" let $y \sqsubseteq_{\mathcal{L}} [\![t]\!]_{\mathcal{P}_{\mathcal{L}}}(x)$. We know for each $a \in A$ and $z \in A^*$ that the statements $[\![a]\!]_{\mathcal{P}_{\mathcal{L}}}(z) \in [\![a]\!]_{\mathcal{PLS}_{\mathcal{L}}}(z)$ and $[\![\overline{a}]\!]_{\mathcal{P}_{\mathcal{L}}}(z) \in [\![\overline{a}]\!]_{\mathcal{PLS}_{\mathcal{L}}}(z)$ hold. From this we also infer $[\![t]\!]_{\mathcal{P}_{\mathcal{L}}}(z) \in [\![t]\!]_{\mathcal{PLS}_{\mathcal{L}}}(z)$ (one can verify this by induction on the length of t). Since $[\![t]\!]_{\mathcal{PLS}_{\mathcal{L}}}(x)$ also is downward closed under $\sqsubseteq_{\mathcal{L}}$, we finally infer that $y \in [\![t]\!]_{\mathcal{PLS}_{\mathcal{L}}}(x)$ holds. ◂

From this equivalence we finally obtain the isomorphism between the transformation monoids of the partially lossy stack $\mathcal{PLS}_{\mathcal{L}}$ and their "read-lossy" semantics $\mathcal{P}_{\mathcal{L}}$.

Theorem 3.3.5. *Let $\mathcal{L}$ be a lossiness alphabet. Then we have $\mathbb{T}(\mathcal{PLS}_{\mathcal{L}}) \cong \mathbb{T}(\mathcal{P}_{\mathcal{L}})$.* ◂

Hence, from now on we are free to call $\mathcal{P}_{\mathcal{L}}$ a pls and its transformation monoid the pls monoid, as well.

3.3.2 Partially Lossy Queues

Next, we want to introduce so-called partially lossy queues. These are queues which are allowed to forget some of their entries specified by a lossiness alphabet. With the help of this data type we are able to model a channel between two network instances with unreliable transmission of the messages. Note that this is a more realistic view on a communication channel than modeling with a reliable queue. Lossy queues (or channels) also received much attention from researchers in the field of verification since the nineties of the last century. This is due to the celebrated result from Abdulla and Jonsson stating that the reachability problem in automata with one or more lossy queues is decidable [AJ96a] (however this problem is not primitive recursive [Scho2, CS08]).

Let $\mathcal{L} = (F, U)$ be a lossiness alphabet. A *partially lossy queue* (*plq*, for short) is the lossy data type $\mathcal{PLQ}_{\mathcal{L}} = (A^*, \Sigma, \Theta, \sqsubseteq_{\mathcal{L}})$ where (A, Σ, Θ) is the data type $\mathcal{Q}_A$ of a queue. The transformation monoid $\mathbb{T}(\mathcal{PLQ}_{\mathcal{L}})$ of a partially lossy queue is called the *partially lossy queue monoid* (or *plq monoid*, for short). We also call the $\mathcal{PLQ}_{\mathcal{L}}$-automata *partially lossy queue automata* (or *plq automata*).

We can also define the "read-lossy" semantics of a plq as follows: consider the data type $\mathcal{Q}_{\mathcal{L}} = (A^*, \Sigma, \Theta)$ where $\Sigma = A \cup \overline{A}$,

$$[\![a]\!]_{\mathcal{Q}_{\mathcal{L}}}(x) := \begin{cases} xa & \text{if } x \in A^* \\ \bot & \text{otherwise}\,, \end{cases} \quad \text{and} \quad [\![\overline{a}]\!]_{\mathcal{Q}_{\mathcal{L}}}(x) := \begin{cases} z & \text{if } x = yaz,\ y \in (F \setminus \{a\})^* \\ \bot & \text{otherwise} \end{cases}$$

for each $a \in A$ and $x \in A^* \cup \{\bot\}$.

Similar to the argumentation in the previous subsection we are able to prove that the "read-lossy" semantics and the "default" semantics of a partially lossy queue are equivalent. In other words, we can prove that the transformation monoids of both data types, $\mathcal{PLQ}_{\mathcal{L}}$ and $\mathcal{Q}_{\mathcal{L}}$, are isomorphic. Due to the similarity we will skip this proof here. However, a full proof can also be found in the author's Master's thesis [Köc16] (for fully lossy queues, only) and in [KKP18] (for arbitrary partially lossy queues).

Theorem 3.3.6. *Let $\mathcal{L}$ be a lossiness alphabet. Then we have $\mathbb{T}(\mathcal{Q}_{\mathcal{L}}) \cong \mathbb{T}(\mathcal{PLQ}_{\mathcal{L}})$.* ◂

Hence, from now on we may also call $\mathcal{Q}_{\mathcal{L}}$ a plq and call its transformation monoid the plq monoid.

While we already know that automata with a reliable queue (i.e., $\mathcal{Q}_{(\varnothing,U)}$-automata) are as powerful as Turing-machines (by [BZ83]), this fact does not hold for fully lossy queue automata (i.e., $\mathcal{Q}_{(F,\varnothing)}$-automata). Note that this is a consequence of the decidable reachability problem and Rice's Theorem. In contrast, plq automata, which are not fully lossy (i.e., we have $U \neq \varnothing$) and have an at least binary alphabet of queue entries (i.e., we have $|A| \geq 2$), are able to simulate Turing-machines:

Theorem 3.3.7. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $U \neq \varnothing$ and $|A| \geq 2$. Then $\mathcal{Q}_{\mathcal{L}}$-automata accept exactly the class of recursively enumerable languages.*

Proof idea. From [BZ83] we know that queue automata are able to simulate Turing-machines. Hence, we only have to simulate queue automata here. So, let $B = \{b_1, \ldots, b_n\}$ be an alphabet and $\mathfrak{Q} = (Q_{\mathfrak{Q}}, \Gamma, \mathcal{Q}_B, I_{\mathfrak{Q}}, c_{\mathfrak{Q}}, \Delta_{\mathfrak{Q}}, F_{\mathfrak{Q}})$ be a queue automaton on B. Additionally, let $\# \in U$ and $a \in A \setminus \{\#\}$ be two distinct letters. We consider the homomorphism $f\colon B^* \to A^*$ which satisfies $f(b_i) := \#^i a \#^{n-i}$ for each $1 \leq i \leq n$. Then it is easy to see that f is injective.

We can construct a $\mathcal{Q}_{\mathcal{L}}$-automaton $\mathfrak{A} = (Q_{\mathfrak{Q}}, \Gamma, \mathcal{Q}_{\mathcal{L}}, I_{\mathfrak{Q}}, c_{\mathfrak{A}}, \Delta_{\mathfrak{A}}, F_{\mathfrak{Q}})$ with

- $c_{\mathfrak{A}} := f(c_{\mathfrak{Q}})$ and
- $\Delta_{\mathfrak{A}} := \{(p, \alpha, f(t), q) \mid (p, \alpha, t, q) \in \Delta_{\mathfrak{Q}}\}$.

Then $(p, v) \xrightarrow{t}_{\mathfrak{Q}} (q, w)$ holds if, and only if, $(p, f(v)) \xrightarrow{f(t)}_{\mathfrak{A}} (q, f(w))$ holds. Note that whenever $\mathfrak{Q}$ fails to read a letter b_i from its queue, $\mathfrak{A}$ also fails to read $f(b_i)$: due to the encoding f, the automaton $\mathfrak{A}$ is able to recognize any loss since each $f(b_i)$ contains exactly n unforgettable #'s and one (possibly forgettable) a.

Hence, we obtain $L(\mathfrak{Q}) = L(\mathfrak{A})$. ◂

Remark 3.3.8. Besides the (partially) lossy queues, there are even more faulty variations of queues. For example, instead of losing entries a faulty queue may also randomly insert entries. In networks with a reliable communication protocol (like TCP) the sender keeps re-sending its messages until it receives an acknowledgment from the receiver. This

may result in undesired multiple copies of a message. Such situation can be modeled by queues with insertion errors, which also have been considered in a few papers like [CFP96, Bou+12]. However, in this thesis we will only focus on the (partially) lossy variant of queues. ⌟

3.4 Distributed Data Types

Until now we have considered only single memories. Now, we want to consider systems having multiple of such storage mechanisms which are distributed in a certain network and which may have some synchronization of their data. Here we focus on data types having only words as their content like queues and stacks.

So, consider a finite number of computers (or processes) each having a storage mechanism represented by a data type. Each of these computers is instructed to do a few special tasks according to their security clearance. For example, computer 1 treats only task a, computer 2 treats the tasks b and c, and computer 3 treats the tasks a and c. In this connection "treating a task" means (e.g.) that the computer may create a new task or finalize a task. Additionally to this, treating of a task α requires the consensus of each computer having the matching security clearance (i.e., all computers which are allowed to treat α). We understand this consensus as follows: we create such task on each computer which should execute a routine on this task. A task can be finalized whenever each computer finished their routines on this task.

Formally, we model our distributed networks of computers as following:

Definition 3.4.1. A *distributed alphabet* is a triple $\mathcal{A} = (A, P, \mathrm{M})$ where A and P are two alphabets (the *tasks* and *processes*, respectively) and $\mathrm{M} \subseteq A \times P$ maps letters to sets of processes such that $a\,\mathrm{M} \neq \varnothing$ and $\mathrm{M}\,i \neq \varnothing$ holds for each $a \in A$ and $i \in P$. ⌟

In our situation P is the set of computers (or data types) in our system, A is the set of tasks, and M describes which computers are allowed to treat tasks. Due to our definition, each task $a \in A$ can be treated by some computers and each computer $i \in P$ is able to treat some tasks.

For a word $w = a_1 \dots a_n$ with $a_1, \dots, a_n \in A$ we write $w\,\mathrm{M} := \bigcup_{1 \leq k \leq n} a_k\,\mathrm{M}$ and for $i \in P$ we write $A_i := \mathrm{M}\,i$.

Recall that the single data types store finite sequences of entries. Then the content of the whole distributed system is a tuple from $\prod_{i \in P} A_i^*$. Whenever we write or read a letter $a \in A$ in our system, we execute this shared action on all processes in $a\,\mathrm{M}$ at the same time. For example, writing an $a \in A$ into distributed queues with content $(w_i)_{i \in P} \in \prod_{i \in P} A_i^*$ yields a new content $(w_i')_{i \in P}$ with $w_i' = w_i a$ for each $i \in a\,\mathrm{M}$ and $w_i' = w_i$ otherwise. Note that computations of our distributed system fails whenever the computation on at least one process fails.

Remark 3.4.2. Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet. There also exist contents of our considered distributed storage which are in some sense inconsistent. For example, consider a content $(w_i)_{i \in P} \in \prod_{i \in P} A_i^*$, two processes $i, j \in P$, and two distinct letters $a, b \in A_i \cap A_j$ such that $w_i = ab$ and $w_j = ba$. Then a distributed system with queue behavior is unable to read a since the process j is blocked by a b. Similarly, we are unable to read b. However, in this case it is possible to write any letter or to read letters from the processes $P \setminus (a\,\mathrm{M} \cup b\,\mathrm{M})$.

Here, we only consider such contents which are "consistent" in the following sense: a distributed content $(w_i)_{i\in P} \in \prod_{i\in P} A_i^*$ is *consistent* if each pair $i, j \in P$ of processes share a common subsequence $\pi_{A_i}(w_j) = \pi_{A_j}(w_i)$. The set of such consistent distributed contents forms a monoid - the so-called trace monoid. We will recapitulate the formal definition and some basic properties of this monoid in the following subsection. ⌟

3.4.1 Intermezzo: Trace Theory

In this subsection we want to define the so-called *trace monoid* and recapitulate a few basic properties of this monoid. The trace monoid was first introduced to computer science by Mazurkiewicz in [Maz77] for modeling concurrent programs and Petri nets. Researchers developed a rich theory on this trace monoid since this publication. A fundamental survey on this topic can be found in Diekert's and Rozenberg's "The Book of Traces" [DR95].

Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet. We say that $a \in A$ is *independent* of $b \in A$ (denoted by $a \parallel b$) if, and only if, $a\,\mathrm{M} \cap b\,\mathrm{M} = \varnothing$ holds. Similarly, for two words $v, w \in A^*$ we write $v \parallel w$ if, and only if, $v\,\mathrm{M} \cap w\,\mathrm{M} = \varnothing$. Then we can construct the *dependence graph* $\mathcal{G}_\mathcal{A} := (A, E_\mathcal{A})$ of $\mathcal{A}$ where $E_\mathcal{A} := \{(a, b) \mid a \nparallel b\}$. For each process $i \in P$ the alphabet A_i is a clique in $\mathcal{G}_\mathcal{A}$.

Let $\approx_\mathcal{A}$ be the least congruence on A^* satisfying $ab \approx_\mathcal{A} ba$ for each pair $a, b \in A$ with $a \parallel b$ (i.e., $(a, b) \in A^2 \setminus E_\mathcal{A}$). This congruence induces the quotient monoid $\mathbb{M}(\mathcal{A}) := A^*/_{\approx_\mathcal{A}}$ which we call the *trace monoid* or the *free partially commutative monoid*. The identity of this monoid is $\varepsilon := [\varepsilon]_{\approx_\mathcal{A}}$. The elements from $\mathbb{M}(\mathcal{A})$ are called *traces* and subsets of $\mathbb{M}(\mathcal{A})$ are called *trace languages*. The *natural epimorphism* is denoted by $\zeta_\mathcal{A} \colon A^* \twoheadrightarrow \mathbb{M}(\mathcal{A}) \colon w \mapsto [w]_{\approx_\mathcal{A}}$. Note that we write $[w]$ instead of $[w]_{\approx_\mathcal{A}}$ whenever the situation is clear. For $a \in A$ we have $[a] = \{a\}$. Due to this fact, we sometimes also write a instead of $[a]$.

Example 3.4.3. Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet with $A = \{a, b, c\}$, $P = \{1, 2, 3\}$, and $\mathrm{M} = \{(a, 1), (a, 3), (b, 2), (c, 2), (c, 3)\}$. Note that this distributed alphabet corresponds to the scenario described above. The dependence graph $\mathcal{G}_\mathcal{A}$ is the following: $a \text{ — } c \text{ — } b$. We have, for example, $abc \approx_\mathcal{A} bac$ but $abc \not\approx_\mathcal{A} acb$. ⌟

Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $B \subseteq A$. The restriction of $\mathcal{A}$ to B is the new distributed alphabet $\mathcal{A}\restriction_B := (B, P, \mathrm{M} \cap (B \times P))$. The *projection* $\pi_B \colon \mathbb{M}(\mathcal{A}) \to \mathbb{M}(\mathcal{A}\restriction_B)$ is the homomorphism induced by the equation

$$\pi_B(a) = \begin{cases} a & \text{if } a \in B \\ \varepsilon & \text{if } a \notin B \end{cases}$$

for each $a \in A$. Note that this homomorphism is well-defined since $a \parallel b$ implies $\pi_B(a) \parallel \pi_B(b)$ for each pair $a, b \in A$ of letters. Additionally, for $i \in P$ by π_i we denote the projection π_{A_i} to all letters from process i. Note that for each $i \in P$ the trace monoid $\mathbb{M}(\mathcal{A}\restriction_{A_i})$ is isomorphic to A_i^* with isomorphism $\zeta_{\mathcal{A}\restriction_{A_i}}$. Hence, we can understand the projection π_i as a map into the free monoid A_i^*. Then we have the following correspondence between projections and commutations:

Theorem 3.4.4 (Projection-Lemma [CL84, CP85]). *Let $\mathcal{A} = (A, P, \mathrm{M})$ and $v, w \in A^*$. Then we have $v \approx_\mathcal{A} w$ if, and only if, $\pi_i(v) = \pi_i(w)$ holds for each $i \in P$.* ◂

Remark 3.4.5. We consider the map $\vec{\pi}\colon A^* \to \prod_{i\in P} A_i^* \colon w \mapsto (\pi_i(w))_{i\in P}$. Then by Theorem 3.4.4 we know that $v \approx_{\mathcal{A}} w$ holds if, and only if, $\vec{\pi}(v) = \vec{\pi}(w)$ holds. In other words, for all traces $\lambda \in \mathbb{M}(\mathcal{A})$ each word $w \in \lambda$ induces the same distributed content $\vec{\pi}(w)$. Hence, we can see w (resp. λ) as a serialization of the content $\vec{\pi}(w)$ of the considered distributed memory. Accordingly, we can see the equivalence class $[w]$ of a word $w \in A^*$ also as its tuple of distributed contents $\vec{\pi}(w)$. Note that the image of $\vec{\pi}$ is the set of all consistent contents in $\prod_{i\in P} A_i^*$. So the trace monoid is isomorphic to a submonoid of $\prod_{i\in P} A_i^*$ - namely, the set of consistent contents. ⌟

From the projection-lemma we obtain several similarities between the trace monoid $\mathbb{M}(\mathcal{A})$ and the free monoid A^*. For example, the trace monoid is cancellative. This means, for each trace $\lambda \in \mathbb{M}(\mathcal{A})$ and each prefix (or suffix, resp.) $\kappa \in \mathbb{M}(\mathcal{A})$ of λ (i.e., we have $\kappa \in \lambda \cdot \mathbb{M}(\mathcal{A})$ resp. $\kappa \in \mathbb{M}(\mathcal{A})\cdot\lambda$), there exists a uniquely defined complementary suffix (or prefix) $\mu \in \mathbb{M}(\mathcal{A})$ with $\lambda = \kappa\mu$ (resp. $\lambda = \mu\kappa$).

However, even if the trace monoid is cancellative it is not *equidivisible*. So, in the free monoid for all words $u, v, w, x \in A^*$ the equation $uv = wx$ is equivalent to the existence of another word $z \in A^*$ with either $uz = w$ and $v = zx$ or $u = wz$ and $zv = x$. In other words, u is a prefix of w and x a suffix of v (or vice versa) and z is the corresponding complementary suffix and prefix. This fact is also known as Levi's lemma [Lev44]. This fact does not hold in the general case in the trace monoid due to its commutations. For example, we consider a distributed alphabet $\mathcal{A} = (A, P, \mathrm{M})$ inducing the dependence graph $a — b — c$. Then for $u = ba$, $v = cb$, $w = bc$, and $x = ab$ we have $uv \approx_{\mathcal{A}} wx$, but neither u is a prefix of w nor vice versa. Indeed, there are words $z_1, z_2, z_3, z_4 \in A^*$ with $u \approx_{\mathcal{A}} z_1z_2$, $v \approx_{\mathcal{A}} z_3z_4$, $w \approx_{\mathcal{A}} z_1z_3$, $x \approx_{\mathcal{A}} z_2z_4$, and $z_2 \parallel z_3$. In our example, this holds for $z_1 = z_4 = b$, $z_2 = a$, and $z_3 = c$. This fact can also be generalized to equations consisting of an arbitrary number of factors. A visualization of this "Levi's lemma for traces" can be found in Figure 3.3.

Theorem 3.4.6 (Levi's lemma for traces [Die90, DM97]). *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\lambda_1, \lambda_2, \ldots, \lambda_m, \kappa_1, \kappa_2, \ldots, \kappa_n \in \mathbb{M}(\mathcal{A})$ be traces. Then we have $\lambda_1\lambda_2\ldots\lambda_m = \kappa_1\kappa_2\ldots\kappa_n$ if, and only if, there are traces $\mu_{i,j} \in \mathbb{M}(\mathcal{A})$ for each $1 \le i \le m$ and $1 \le j \le n$ such that the following properties hold:*

(a) $\lambda_i = \mu_{i,1}\mu_{i,2}\ldots\mu_{i,n}$ *for each* $1 \le i \le m$,

(b) $\kappa_j = \mu_{1,j}\mu_{2,j}\ldots\mu_{m,j}$ *for each* $1 \le i \le n$, *and*

(c) $\mu_{i,j} \parallel \mu_{k,\ell}$ *for each* $1 \le i < k \le m$ *and* $1 \le \ell < j \le n$ *(hence,* $\mu_{i,j}\mu_{k,\ell} = \mu_{k,\ell}\mu_{i,j}$*).* ◂

For $w \in A^*$ we write $|[w]_{\approx_{\mathcal{A}}}| := |w|$ and $|[w]_{\approx_{\mathcal{A}}}|_B := |\pi_B([w]_{\approx_{\mathcal{A}}})| = |w|_B$ for the number of occurrences of letters from B in w.

By $\mathrm{Isolated}(\mathcal{A}) := \{a \in A \mid \forall b \in A \setminus \{a\} \colon a \parallel b\} = \{a \in A \mid a\, E_{\mathcal{A}} = \{a\}\}$ we denote the set of *isolated nodes* in $\mathcal{G}_{\mathcal{A}}$. For $a \in \mathrm{Isolated}(\mathcal{A})$ it is easy to see that $a\mathbb{M}(\mathcal{A}) = \mathbb{M}(\mathcal{A})a$ holds.

Let $\lambda \in \mathbb{M}(\mathcal{A})$ be a trace. Then the *induced alphabet* of λ is $\mathrm{Alph}(\lambda) := \{a \in A \mid \lambda \in \mathbb{M}(\mathcal{A})a\mathbb{M}(\mathcal{A})\}$, i.e., $\mathrm{Alph}(\lambda)$ is the set of all letters in λ. The trace λ is *connected* if $\mathrm{Alph}(\lambda)$ is a connected set in $\mathcal{G}_{\mathcal{A}}$. A trace language $L \subseteq \mathbb{M}(\mathcal{A})$ is *connected* if each trace $\lambda \in L$ is connected.

	κ_1	$\cdots$	κ_{j-1}	κ_j	κ_{j+1}	$\cdots$	κ_n
λ_1	$\mu_{1,1}$	$\cdots$	$\mu_{1,j-1}$	$\mu_{1,j}$	$\mu_{1,j+1}$	$\cdots$	$\mu_{1,n}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\iddots$	$\vdots$
λ_{i-1}	$\mu_{i-1,1}$	$\cdots$	$\mu_{i-1,j-1}$	$\mu_{i-1,j}$	$\mu_{i-1,j+1}$	$\cdots$	$\mu_{i-1,n}$
λ_i	$\mu_{i,1}$	$\cdots$	$\mu_{i,j-1}$	$\boldsymbol{\mu_{i,j}}$	$\mu_{i,j+1}$	$\cdots$	$\mu_{i,n}$
λ_{i+1}	$\mu_{i+1,1}$	$\cdots$	$\mu_{i+1,j-1}$	$\mu_{i+1,j}$	$\mu_{i+1,j+1}$	$\cdots$	$\mu_{i+1,n}$
$\vdots$	$\vdots$	$\iddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
λ_m	$\mu_{m,1}$	$\cdots$	$\mu_{m,j-1}$	$\mu_{m,j}$	$\mu_{m,j+1}$	$\cdots$	$\mu_{m,n}$

Figure 3.3. Visualization of Levi's lemma for traces. Row-wise concatenation of the $\mu_{i,j}$ results in λ_i (this is property (a)) and concatenating them column-wise yields κ_j (this is property (b)). Additionally, the entries in the red-shaded cells are independent of $\mu_{i,j}$ (this is property (c)).

Finally, a *distributed data type* over the distributed alphabet $\mathcal{A}$ is a data type $\mathcal{D} = (U, \Sigma, \Theta)$ with $U = \mathbb{M}(\mathcal{A})$.

3.4.2 Distributed Stacks

Distributed stacks are the distributed version of (reliable) stacks. According to our interpretation of distributed data types, we consider systems consisting of multiple instances of a stack. Between those stacks we have a special synchronization mechanism in the following sense: any entry $a \in A$ is associated to a subset $a\,\mathrm{M} \subseteq P$ of the stacks (or processes). Whenever we write or read a letter a we do this action at the same time on each stack $i \in a\,\mathrm{M}$. With the help of such distributed stacks we are able to model concurrent programs with recursive function calls. Such systems have received an increased attention in recent years [Heu+10, LN11, CGK12, BKM17].

Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet. A *distributed stack* (or *distributed pushdown*) is the distributed data type $\mathcal{P}_{\mathcal{A}} := (\mathbb{M}(\mathcal{A}), \Sigma, \Theta)$ where $\Sigma = A \cup \overline{A}$,

$$[\![a]\!](\lambda) := \begin{cases} a\lambda & \text{if } \lambda \in \mathbb{M}(\mathcal{A}) \\ \bot & \text{otherwise,} \end{cases} \quad \text{and} \quad [\![\overline{a}]\!](\lambda) := \begin{cases} \kappa & \text{if } \lambda = a\kappa \\ \bot & \text{otherwise} \end{cases}$$

holds for each $a \in A$ and $\lambda \in \mathbb{M}(\mathcal{A}) \cup \{\bot\}$. Note that all transformations are well-defined due to the cancellation property of the trace monoid $\mathbb{M}(\mathcal{A})$. One can observe that the definitions of distributed stacks $\mathcal{P}_{\mathcal{A}}$ and (reliable) stacks $\mathcal{P}_A$ only differ in their concrete universe. So, distributed stacks have a trace monoid as their universe $\mathbb{M}(\mathcal{A})$ instead of the free monoid A^*. Due to this observation, a distributed stack consists of $|P|$ many reliable stacks on the alphabets A_i (where $i \in P$) which are synchronized according to the underlying distributed alphabet.

Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet. A *distributed pushdown automaton* is a $\mathcal{P}_{\mathcal{A}}$-automaton. Note that if $\mathcal{G}_{\mathcal{A}}$ is the disjoint union of cliques then a distributed pushdown automaton over $\mathcal{A}$ is essentially a multi-pushdown automaton. It is well-known that these pushdown automata with at least two stacks are as powerful as Turing-machines and, hence, accept exactly the recursively enumerable languages (cf. [HMU01, Section 8.5.2]).

If $\mathcal{G}_{\mathcal{A}}$ is a complete graph then we have $\mathbb{M}(\mathcal{A}) \cong A^*$ and, hence, $\mathcal{P}_{\mathcal{A}} \cong \mathcal{P}_A$. In this case the distributed pushdown automata over $\mathcal{A}$ are essentially (non-distributed) pushdown automata. Therefore, those automata accept exactly the class of context-free languages.

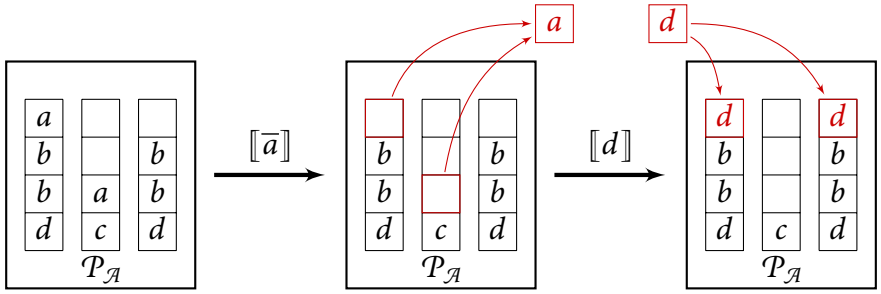

Figure 3.4. A visualization of a distributed stack with content $[acbbd]$ applying the transformation $[\![\overline{a}d]\!]$.

3.4.3 Distributed Queues

Similar to the distributed stacks we can also define distributed queues. These are multiple (reliable) queues with the synchronization between the single queues as described in the previous subsection. Such systems had been considered for example by Hutagalung et al. in [Hut+18]. Note that this is a generalization of multiple queues. With the help of such multi-queues we are able to model systems consisting of a finite number of network instances and some (reliable) communication channels between those instances. Automata with multiple queues are called multi-queue automata or communicating automata and were studied in many papers in the field of verification, for example in [Boi+97, Bol06, KM21].

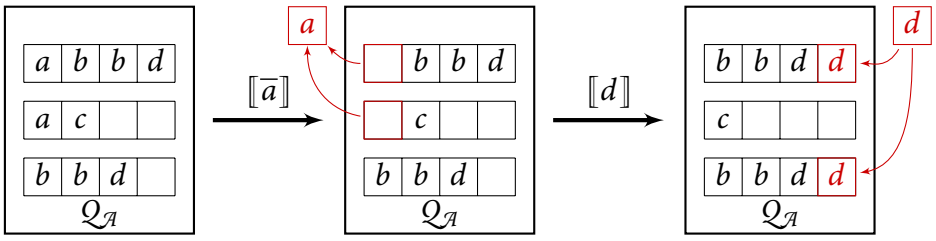

Figure 3.5. A visualization of a distributed queue with content $[acbbd]$ applying the transformation $[\![\overline{a}d]\!]$.

Formally, a *distributed queue* is the distributed data type $\mathcal{Q}_\mathcal{A} := (\mathbb{M}(\mathcal{A}), \Sigma, \Theta)$ for a distributed alphabet $\mathcal{A} = (A, P, \mathrm{M})$ where $\Sigma = A \cup \overline{A}$,

$$[\![a]\!](\lambda) := \begin{cases} \lambda a & \text{if } \lambda \in \mathbb{M}(\mathcal{A}) \\ \bot & \text{otherwise,} \end{cases} \quad \text{and} \quad [\![\overline{a}]\!](\lambda) := \begin{cases} \kappa & \text{if } \lambda = a\kappa \\ \bot & \text{otherwise} \end{cases}$$

holds for each $a \in A$ and $\lambda \in \mathbb{M}(\mathcal{A}) \cup \{\bot\}$. Note that all transformations of $\mathcal{Q}_\mathcal{A}$ are well-defined due to the cancellation property of the trace monoid. Again, we can observe that the only difference of the definitions of a distributed queue $\mathcal{Q}_\mathcal{A}$ and a reliable queue $\mathcal{Q}_A$ is their universe. We can understand a distributed queue as $|P|$ many reliable queues with alphabets A_i (for $i \in P$). Therefore, we can also observe the following:

Observation 3.4.7. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a dependence alphabet, $t \in \Sigma^*$ be an action sequence, and $\kappa, \lambda \in \mathbb{M}(\mathcal{A})$ with $[\![t]\!](\kappa) = \lambda \neq \bot$. Then we have $\kappa \cdot [w] = [r] \cdot \lambda$ where $w = \pi_A(t)$ and $r = \pi_{\overline{A}}(t)$.* ◂

Note that if $\mathcal{G}_{\mathcal{A}}$ is the disjoint union of cliques then $\mathcal{Q}_{\mathcal{A}}$ essentially consists of multiple independent queues. If $\mathcal{G}_{\mathcal{A}}$ is a complete graph then we have $\mathbb{M}(\mathcal{A}) \cong A^*$ and, hence, $\mathcal{Q}_{\mathcal{A}} \cong \mathcal{Q}_A$ in this case. Anyway, *distributed queue automata* (these are the $\mathcal{Q}_{\mathcal{A}}$-automata) are Turing-complete and, therefore, accept exactly the recursively enumerable languages.

Part I
Properties of Transformation Monoids

Chapter 4

Behavioral Equivalence

4.1 Introduction

In Chapter 3 we have introduced the notion of data types and defined several examples with the help of concrete semantics of the basic actions of these data types. So, for example the write action a of a (reliable) stack corresponds to prepending the letter a to the content of the stack while the same action on a queue corresponds to appending this letter to the queue's content. In literature, sometimes data types are also defined with the help of several axioms. In other words, we describe the semantics of a data type with the help of one or more equations. For example, we can find such kind of equations for reliable stacks in (e.g.) [Kam09, RK09], for reliable queues in [HKZ17], for (partially) lossy queues in [Köc16, KKP18], and for some further, more complicated data types in [Zet16].

With the help of such axioms we are able to find out certain algebraic properties of the transformation monoid $\mathbb{T}(\mathcal{D})$ of a data type $\mathcal{D}$. For example, these equations help to understand the semantics of the composition of multiple transformations. The axioms also help when analyzing the cancellation properties or Green's relations of the transformation monoid. The knowledge of these algebraic properties turns out to be very helpful later in this thesis when studying the languages in the transformation monoid and the reachability problems of $\mathcal{D}$-automata.

In this chapter we will study the algebraic properties of the transformation monoid of reliable, partially lossy, and distributed stacks and queues. We do this with the help of a special congruence on the sequences of basic actions - the so called behavioral equivalence. In this connection, we say that two action sequences behave equivalent if, and only if, they have the same semantics. We can see then that the transformation monoid $\mathbb{T}(\mathcal{D})$ of the data type $\mathcal{D}$ is essentially the quotient of the sequences of basic actions under the behavioral equivalence. When investigating the properties of the transformation monoid and the behavioral equivalence our proof strategy always is the same: we start with proving some simple equations that hold in the transformation monoid. From these equations we gain a confluent and terminating semi-Thue system on the sequences of basic actions. With the help of this system we are able to represent the transformations of the considered data type by unique, irreducible words over the alphabet Σ, which we call the *normal forms*. With the help of these normal forms we will show that our equations are complete axiomatic descriptions of our data types. Additionally, we will derive cancellation properties and characterizations of Green's relations from these normal forms.

4.2 Definition

Let $\mathcal{D} = (U, \Sigma, \Theta)$ be a data type. We say that two sequences of actions $s, t \in \Sigma^*$ *behave equivalently* if, and only if, $[\![s]\!] = [\![t]\!]$ holds. In other words, s and t behave equivalent, if the application of s to any content $x \in U$ leads to the same content as when applying t to x. We write $s \equiv_{\mathcal{D}} t$ in this case. We obtain an equivalence relation $\equiv_{\mathcal{D}}$ which we call the *behavioral equivalence*. Note that we write $\equiv$ instead of $\equiv_{\mathcal{D}}$ whenever the situation is clear. We also write $[t]$ instead of $[t]_{\equiv_{\mathcal{D}}}$ for the equivalence class of an action sequence $t \in \Sigma^*$ whenever the situation is clear.

The following properties of the behavioral equivalence are well-known:

Fact 4.2.1. *Let $\mathcal{D} = (U, \Sigma, \Theta)$ be a data type. The following statements hold:*

(1) The relation $\equiv_{\mathcal{D}}$ is a congruence on the free monoid Σ^*.

(2) $\Sigma^*/_{\equiv_{\mathcal{D}}} \cong \mathbb{T}(\mathcal{D})$. ◂

The isomorphism from the second statement in Fact 4.2.1 is $\Sigma^*/_{\equiv_{\mathcal{D}}} \to \mathbb{T}(\mathcal{D}) : [t] \mapsto [\![t]\!]$ for each $t \in \Sigma^*$. Hence, we can use the equivalence class $[t]$ of t and its semantics $[\![t]\!]$ synonymously.

We denote the *natural epimorphism* on $\equiv_{\mathcal{D}}$ by $\eta_{\mathcal{D}} : \Sigma^* \twoheadrightarrow \Sigma^*/_{\equiv_{\mathcal{D}}} : t \mapsto [t]$.

4.3 Recapitulation: Reliable Stacks

In this section we want to recall basic properties of the behavioral equivalence and of the transformation monoid of a (reliable) stack. It is well-known that the transformation monoid is the so-called polycyclic semigroup [NP70]. So the right-inverse element of the transformation $[\![a]\!]$ is $[\![\overline{a}]\!]$ for any $a \in A$. However, $[\![\overline{a}]\!]$ is not the left-inverse element of $[\![a]\!]$. Consequentially, we also obtain $a\overline{a} \equiv \varepsilon$ but not $\overline{a}a \equiv \varepsilon$. Moreover, for two distinct letters $a, b \in A$ we obtain

$$[\![a\overline{b}]\!](x) = [\![\overline{b}]\!](ax) = \bot \tag{4.1}$$

for each stack content $x \in A^*$. This means that $[\![a\overline{b}]\!]$ is a constant function mapping any input to the error state $\bot$. This also implies

$$[\![s\, a\overline{b}\, t]\!](x) = [\![t]\!]([\![a\overline{b}]\!]([\![s]\!](x))) = [\![t]\!](\bot) = \bot$$

for all action sequences $s, t \in \Sigma^*$ and each content $x \in A^* \cup \{\bot\}$. Hence, the transformation $[\![a\overline{b}]\!]$ is an annihilating (or zero-)element of the transformation monoid $\mathbb{T}(\mathcal{P}_A)$ wrt. composition. For simplicity we introduce a new action name $\lightning \notin \Sigma$ which is the name of this annihilating element (i.e., we have $[\![\lightning]\!](x) = \bot$ for any $x \in A^* \cup \{\bot\}$). We also extend our alphabet of basic actions to $\Sigma_{\lightning} := \Sigma \cup \{\lightning\}$. Since $\lightning$ is just a synonym of the action sequence $a\overline{b}$ we have $\Sigma_{\lightning}^*/_{\equiv} \cong \Sigma^*/_{\equiv} \cong \mathbb{T}(\mathcal{P}_A)$.

Note that this annihilating element exists only if our stack is able to store at least two distinct entries. Whenever the stack is essentially a counter (i.e., we have $|A| = 1$), there is no such annihilating element $[\![\lightning]\!]$.

All in all, we obtain the following simple equations that hold in the behavioral equivalence of a stack.

Lemma 4.3.1. *Let A be an alphabet, $a, b \in A$ be two letters, and $\alpha \in \Sigma_{\perp\!\!\!\perp}$ be an action. Then the following equations hold:*

(1) $a\overline{a} \equiv \varepsilon$,

(2) $a\overline{b} \equiv \perp\!\!\!\perp$ *if* $a \neq b$, *and*

(3) $\alpha\perp\!\!\!\perp \equiv \perp\!\!\!\perp\alpha \equiv \perp\!\!\!\perp$. ◂

Our next aim is to show that these equations fully describe the behavioral equivalence of a stack. To this end, we consider the semi-Thue system $\mathfrak{S}_A$ which arises from ordering the equations in Lemma 4.3.1 from left to right:

(1) $a\overline{a} \to \varepsilon$,

(2) $a\overline{b} \to \perp\!\!\!\perp$ if $a \neq b$, and

(3) $\alpha\perp\!\!\!\perp \to \perp\!\!\!\perp$ and $\perp\!\!\!\perp\alpha \to \perp\!\!\!\perp$

for each $a, b \in A$ and $\alpha \in \Sigma_{\perp\!\!\!\perp}$. This semi-Thue system is finite, monadic[iii], and terminating[iv]. It is also locally confluent[v]: let $a, b \in A$. Then we have

$$\perp\!\!\!\perp \overset{(3)}{\Longleftarrow}{}^{*}_{\mathfrak{S}_A} \alpha\perp\!\!\!\perp \overset{(1)/(2)}{\Longleftarrow}_{\mathfrak{S}_A} a\overline{b}\perp\!\!\!\perp \overset{(3)}{\Longrightarrow}_{\mathfrak{S}_A} a\perp\!\!\!\perp \overset{(3)}{\Longrightarrow}_{\mathfrak{S}_A} \perp\!\!\!\perp \quad \text{and}$$
$$\perp\!\!\!\perp \overset{(3)}{\Longleftarrow}{}^{*}_{\mathfrak{S}_A} \perp\!\!\!\perp\alpha \overset{(1)/(2)}{\Longleftarrow}_{\mathfrak{S}_A} \perp\!\!\!\perp a\overline{b} \overset{(3)}{\Longrightarrow}_{\mathfrak{S}_A} \perp\!\!\!\perp\overline{b} \overset{(3)}{\Longrightarrow}_{\mathfrak{S}_A} \perp\!\!\!\perp$$

where $\alpha \in \{\varepsilon, \perp\!\!\!\perp\}$ with $a\overline{b} \to \alpha$. Since these are the only relevant overlaps of two rules from $\mathfrak{S}_A$, it is locally confluent. From the termination property we finally infer confluence. Hence, for each action sequence $t \in \Sigma^*_{\perp\!\!\!\perp}$ we obtain a uniquely defined and irreducible word $\mathrm{nf}(t)$ (the so-called *normal form* of t) with $t \Rightarrow^*_{\mathfrak{S}_A} \mathrm{nf}(t)$. We denote the set of all normal forms by $\mathrm{NF}_{\mathcal{P}_A}$. From the shape of the rules of $\mathfrak{S}_A$ we obtain

$$\mathrm{NF}_{\mathcal{P}_A} = \overline{A}^* A^* \cup \{\perp\!\!\!\perp\}. \tag{4.2}$$

It is well-known that this normal form is unique in any equivalence class of $\equiv$:

Theorem 4.3.2. *Let A be an alphabet and $s, t \in \Sigma^*_{\perp\!\!\!\perp}$. Then we have $s \equiv t$ if, and only if,* $\mathrm{nf}(s) = \mathrm{nf}(t)$. ◂

Hence, the equations from Lemma 4.3.1 fully describe $\equiv$. From this theorem we obtain that the map $\mathbb{T}(\mathcal{P}_A) \to \mathrm{NF}_{\mathcal{P}_A} : [\![t]\!] \mapsto \mathrm{nf}(t)$ is well-defined. So, from now on we can also write

[iii] A semi-Thue system is *monadic* if for each rule $\ell \to r$ we have $|\ell| > |r|$.

[iv] A semi-Thue system is *terminating* (or *noetherian*) if it has no infinite derivations.

[v] A semi-Thue system over Γ is *confluent* if for each $u, v, w \in \Gamma^*$ with $u \Leftarrow^* v \Rightarrow^* w$ there is $x \in \Gamma^*$ with $u \Rightarrow^* x \Leftarrow^* w$. It is *locally confluent* if $u \Leftarrow v \Rightarrow w$ implies $u \Rightarrow^* x \Leftarrow^* w$.

$\mathrm{nf}(\llbracket t \rrbracket) := \mathrm{nf}(t)$ for each action sequence $t \in \Sigma_{\perp\!\!\!\perp}^*$. Additionally, we obtain from this theorem and from Equation (4.2) for each action sequence $t \in \Sigma_{\perp\!\!\!\perp}^*$ a uniquely defined action sequence $\mathrm{nf}(t)$ which is in some sense "simple" and behaves equivalently to t.

Finally, we want to give a characterization of the normal form of the composition of two action sequences. To this end, let $s, t \in \Sigma_{\perp\!\!\!\perp}^*$ be two action sequences. W.l.o.g., we can assume that s and t are in normal form. Then if $s = \perp\!\!\!\perp$ or $t = \perp\!\!\!\perp$ holds, we also know that $st \equiv \perp\!\!\!\perp$ holds. Otherwise we have $s = \overline{r_1} w_1$ and $t = \overline{r_2} w_2$ for words $r_1, r_2, w_1, w_2 \in A^*$. Then we can possibly apply equation (1) of Lemma 4.3.1 to the subsequence $w_1 \overline{r_2}$. So, iterated application of this equation removes read actions $\overline{x} \in \overline{A}^*$ from r_2 and the corresponding write actions (in reverse order) from w_1. When this iteration ends we obtain two words $w_1', r_2' \in A^*$ with $st \equiv \overline{r_1} w_1' \overline{r_2'} w_2$. If one of these two words is empty, the word $\overline{r_1} w_1' \overline{r_2'} w_2$ is in normal form and we are done. Otherwise, there are two distinct letters $a, b \in A^*$ with $w_1' \in A^* a$ and $r_2' \in bA^*$. Then application of equations (2) and (3) of Lemma 4.3.1 yields $st \equiv \perp\!\!\!\perp$. All in all, we obtain the following characterization:

Theorem 4.3.3 ([Gil96, Lemma 7.1]). *Let A be an alphabet, $r_1, r_2, w_1, w_2 \in A^*$, and $x \in A^*$ is the longest prefix of r_2 such that x^{R} is a suffix of w_1. Then the following statements hold:*

(1) if $w_1 = u x^{\mathrm{R}}$ and $r_2 = x$ then we have $\mathrm{nf}(\overline{r_1} w_1 \overline{r_2} w_2) = \overline{r_1} u w_2$,

(2) if $w_1 = x^{\mathrm{R}}$ and $r_2 = xv$ then we have $\mathrm{nf}(\overline{r_1} w_1 \overline{r_2} w_2) = \overline{r_1 v} w_2$, and

(3) if $w_1 \neq x^{\mathrm{R}}$ and $r_2 \neq x$ then we have $\mathrm{nf}(\overline{r_1} w_1 \overline{r_2} w_2) = \perp\!\!\!\perp$. ◂

4.4 Partially Lossy Stacks

Now we want to generalize the results from the previous section to partially lossy stacks. Again, we want to find a semi-Thue system and define normal forms of the equivalence classes with the help of this system.

Let $\mathcal{L} = (F, U)$ be a lossiness alphabet. First, assume $|A| = 1$ then we know that $\mathcal{P}_{\mathcal{L}}$ is a reliable stack with one stack symbol, i.e., it is essentially a single counter. Note that it does not matter whether $F = A$ or $U = A$ holds in this case. Since we already considered the transformation monoid of this data type in the previous section, we assume $|A| \geq 2$ from now on. So, let $a, b \in A$ be two distinct letters. The transformation $\llbracket \overline{a} \rrbracket$ is still the right-inverse of $\llbracket a \rrbracket$ but not its left-inverse. However, the effect of $\llbracket a\overline{b} \rrbracket$ depends on whether we have $a \in F$ or $a \in U$. First, assume $a \in F$. Then we have for any $x \in A^*$

$$\llbracket a\overline{b} \rrbracket(x) = \llbracket \overline{b} \rrbracket(ax) = \llbracket \overline{b} \rrbracket(x) \tag{4.3}$$

since we have $a \in F \setminus \{b\}$. In other words, we observe $a\overline{b} \equiv \overline{b}$ in this case. Otherwise, if $a \in U$ holds, we know for any $x \in A^*$

$$\llbracket a\overline{b} \rrbracket(x) = \llbracket \overline{b} \rrbracket(ax) = \bot\,. \tag{4.4}$$

Note that this is similar to the situation in reliable stacks (cf. Equation (4.2)). Hence, $\llbracket a\overline{b} \rrbracket$ is an annihilating or zero-element of $\mathbb{T}(\mathcal{P}_{\mathcal{L}})$. Again, we introduce the new action name

$\perp\!\!\!\perp \notin \Sigma$ with $[\![\perp\!\!\!\perp]\!](x) = \perp$ (for each $x \in A^*$) and extend our alphabet of action names Σ to $\Sigma_{\perp\!\!\!\perp} := \Sigma \cup \{\perp\!\!\!\perp\}$. Since we require $a \in U$ for the existence of this annihilating element $[\![\perp\!\!\!\perp]\!]$, there is no such element in the transformation monoid of fully lossy stacks (i.e., whenever $U = \varnothing$ holds). In this case we assume $\Sigma_{\perp\!\!\!\perp} = \Sigma$.

4.4.1 Normal Forms

Again, we start our study of the pls monoid with a list of simple equations that hold in the behavioral equivalence. Later, in Theorem 4.4.2, we will see that this list is a full characterization of this congruence.

Lemma 4.4.1. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $a, b \in A$, and $\alpha \in \Sigma_{\perp\!\!\!\perp}$. Then the following equations hold:*

(1) $a\overline{a} \equiv \varepsilon$,

(2) $a\overline{b} \equiv \overline{b}$ *if* $a \in F \setminus \{b\}$,

(3) $a\overline{b} \equiv \perp\!\!\!\perp$ *if* $a \in U \setminus \{b\}$, *and*

(4) $\alpha \perp\!\!\!\perp \equiv \perp\!\!\!\perp \alpha \equiv \perp\!\!\!\perp$.

As we can see, equation (1) corresponds to the fact that $[\![\overline{a}]\!]$ is the right-inverse of $[\![a]\!]$ for each $a \in A$. The equations (2) and (3) correspond to the (partial) lossiness behavior of $\mathcal{P}_{\mathcal{L}}$ and equation (4) states that $[\![\perp\!\!\!\perp]\!]$ is annihilating.

Proof. First we prove statement (1). To this end, let $a \in A$ and $x \in A^*$. Then we have:

$$[\![a\overline{a}]\!](x) = [\![\overline{a}]\!](ax) = x = [\![\varepsilon]\!](x)\,.$$

Hence, we have $[\![a\overline{a}]\!] = [\![\varepsilon]\!]$ implying $a\overline{a} \equiv \varepsilon$.

Towards the proof of (2) let $a \in F$, $b \in A \setminus \{a\}$, and $x \in A^*$. First, assume $[\![a\overline{b}]\!](x) \neq \perp$. Then there are $y \in (F \setminus \{b\})^*$ and $z \in A^*$ with $ax = ybz$. Since $a \neq b$ and $a \in F$, the word y is not empty. Hence, there is $y' \in (F \setminus \{b\})^*$ with $y = ay'$. Then we have $x = y'bz$ and

$$[\![a\overline{b}]\!](x) = [\![\overline{b}]\!](ax) = [\![\overline{b}]\!](ay'bz) = z = [\![\overline{b}]\!](y'bz) = [\![\overline{b}]\!](x)\,.$$

Conversely, if $[\![\overline{b}]\!](x) \neq \perp$ there are $y \in (F \setminus \{b\})^*$ and $z \in A^*$ with $x = ybz$. Then we have

$$[\![\overline{b}]\!](x) = [\![\overline{b}]\!](ybz) = z = [\![\overline{b}]\!](aybz) = [\![a\overline{b}]\!](x)\,.$$

Statement (3) was already proved in Equation (4.4). Finally, item (4) holds since $[\![s \perp\!\!\!\perp t]\!](x) = \perp = [\![\perp\!\!\!\perp]\!](x)$ holds for each $x \in A^* \cup \{\perp\}$ and $s, t \in \Sigma_{\perp\!\!\!\perp}^*$. ◂

Next, we order the equations from Lemma 4.4.1 from left to right and obtain a semi-Thue system $\mathfrak{S}_{\mathcal{L}}$ with the following rules:

(1) $a\overline{a} \to \varepsilon$,

(2) $a\overline{b} \to \overline{b}$ if $a \in F \setminus \{b\}$,

(3) $a\overline{b} \to \perp\!\!\!\perp$ if $a \in U \setminus \{b\}$,

(4) $\alpha \perp\!\!\!\perp \to \perp\!\!\!\perp$, and $\perp\!\!\!\perp \alpha \to \perp\!\!\!\perp$

where $a, b \in A$ are two letters and $\alpha \in \Sigma_{\perp\!\!\!\perp}$. Similar to the semi-Thue system $\mathfrak{S}_A$ the system $\mathfrak{S}_{\mathcal{L}}$ is finite, monadic, terminating, and confluent (the proofs of these properties are very similar to the reliable case). Hence, for each word $t \in \Sigma_{\perp\!\!\!\perp}^*$ there is a uniquely defined, irreducible word which we call $\mathrm{nf}(t)$ (the so-called *normal form*) and which satisfies $t \Rightarrow_{\mathfrak{S}_{\mathcal{L}}}^* \mathrm{nf}(t)$. The set of all words in normal form is denoted by $\mathrm{NF}_{\mathcal{P}_{\mathcal{L}}}$. Due to the rules of our semi-Thue system $\mathfrak{S}_{\mathcal{L}}$ we obtain

$$\mathrm{NF}_{\mathcal{P}_{\mathcal{L}}} = \overline{A}^* A^* \cup \{\perp\!\!\!\perp\}\,.$$

We prove next, that the normal form is unique in any equivalence class. Hence, the equations from Lemma 4.4.1 fully describe the behavioral equivalence.

Theorem 4.4.2. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $s, t \in \Sigma_{\perp\!\!\!\perp}^*$. Then we have*

$$s \equiv t \iff \mathrm{nf}(s) = \mathrm{nf}(t)\,.$$

Proof. First we prove the implication "⇒". To this end, let $s, t \in \Sigma_{\perp\!\!\!\perp}^*$. If $\mathrm{nf}(s) \neq \perp\!\!\!\perp$ then there are $r_1, w_1 \in A^*$ with $\mathrm{nf}(s) = \overline{r_1} w_1$. Then we have

$$[\![t]\!](r_1) = [\![s]\!](r_1) = [\![\overline{r_1} w_1]\!](r_1) = [\![w_1]\!](\varepsilon) = {w_1}^{\mathrm{R}} \neq \bot$$

and, hence, $t \not\equiv \perp\!\!\!\perp$ implying the existence of $r_2, w_2 \in A^*$ with $\mathrm{nf}(t) = \overline{r_2} w_2$. We prove next $r_1 = r_2$. By the calculation above we have $\bot \neq [\![t]\!](r_1) = [\![\overline{r_2} w_2]\!](r_1)$. This implies $|r_2| \leq |r_1|$. By symmetry, we have $|r_1| = |r_2|$ implying $r_1 = r_2$ since $[\![\overline{r_2}]\!](r_1) \neq \bot$.

Next, we have to prove $w_1 = w_2$. This equation holds since

$${w_1}^{\mathrm{R}} = [\![\overline{r_1} w_1]\!](r_1) = [\![s]\!](r_1) = [\![t]\!](r_1) = [\![\overline{r_1} w_2]\!](r_1) = {w_2}^{\mathrm{R}}\,.$$

This implies $\perp\!\!\!\perp \neq \mathrm{nf}(s) = \mathrm{nf}(t)$.

Note that by symmetry we obtain $\mathrm{nf}(s) \neq \perp\!\!\!\perp$ if, and only if, $\mathrm{nf}(t) \neq \perp\!\!\!\perp$ holds. Hence, if $\mathrm{nf}(s) = \perp\!\!\!\perp$ holds, we also obtain $\mathrm{nf}(t) = \perp\!\!\!\perp = \mathrm{nf}(s)$.

Finally, the converse implication "⇐" holds since $s \equiv \mathrm{nf}(s) = \mathrm{nf}(t) \equiv t$ which holds due to Lemma 4.4.1. ◂

Again, we obtain from this theorem that the map $\mathbb{T}(\mathcal{P}_{\mathcal{L}}) \to \mathrm{NF}_{\mathcal{P}_{\mathcal{L}}} : [\![t]\!] \mapsto \mathrm{nf}(t)$ is well-defined. We can also write $\mathrm{nf}([\![t]\!])$ for the normal form $\mathrm{nf}(t)$ of the action sequence $t \in \Sigma_{\perp\!\!\!\perp}^*$.

4.4.2 Cancellation

Let $\mathcal{L} = (F, U)$ be a lossiness alphabet. Then we can see that $\mathbb{T}(\mathcal{P}_{\mathcal{L}})$ is neither left- nor right-cancellative: let $a \in A$ be a letter then we have $\overline{a}\overline{a}aa \not\equiv \overline{a}a$ by the equations from Lemma 4.4.1. Nevertheless, we obtain the following equations:

$$\overline{a}\overline{a}aa \cdot \overline{a}a \equiv \overline{a}\overline{a}aa \equiv \overline{a}\overline{a}aa \cdot \overline{a}\overline{a}aa \equiv \overline{a}a \cdot \overline{a}\overline{a}aa\,.$$

Note that this property is inherited from the bicyclic semigroup (resp. the transformation monoid of a counter $\mathbb{T}(C_1) \cong \mathbb{T}(\mathcal{P}_{\{a\}})$).

However, the pls monoids are at least cancellative with the following restriction:

Corollary 4.4.3. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $s, t \in \Sigma_{\shortparallel}^*$, and $x, y \in A^*$. Then $\overline{x}sy \equiv \overline{x}ty$ implies $s \equiv t$.* ◂

4.4.3 Green's Relations

Next, we consider Green's relations in the pls monoid. So, let $\mathcal{L} = (F, U)$ be a lossiness alphabet. First, we consider the case $|A| = 1$. In other words, $\mathbb{T}(\mathcal{P}_\mathcal{L})$ is the bicyclic semigroup. For this monoid the following results are well-known: we have $[\![\overline{a}^k a^\ell]\!] \,\mathrm{L}\, [\![\overline{a}^m a^n]\!]$ if, and only if, $\ell = n$ (where $A = \{a\}$). Similarly, we have $[\![\overline{a}^k a^\ell]\!] \,\mathrm{R}\, [\![\overline{a}^m a^n]\!]$ if, and only if, $k = m$ holds [How95]. Now, let $|A| \geq 2$. We will see in this subsection that Green's relations of $\mathbb{T}(\mathcal{P}_\mathcal{L})$ have similar properties. To this end, we prove the following properties of the principal ideals[vi] of $\mathbb{T}(\mathcal{P}_\mathcal{L})$.

Lemma 4.4.4. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $r, w \in A^*$. Then the following statements hold:*

(1) $\mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![\overline{r}w]\!] = \mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![w]\!]$,

(2) $[\![\overline{r}w]\!] \cdot \mathbb{T}(\mathcal{P}_\mathcal{L}) = [\![\overline{r}]\!] \cdot \mathbb{T}(\mathcal{P}_\mathcal{L})$,

(3) $\mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![\overline{r}w]\!] \cdot \mathbb{T}(\mathcal{P}_\mathcal{L}) = \mathbb{T}(\mathcal{P}_\mathcal{L})$, *and*

(4) $\mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![\shortparallel]\!] \cdot \mathbb{T}(\mathcal{P}_\mathcal{L}) = \mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![\shortparallel]\!] = [\![\shortparallel]\!] \cdot \mathbb{T}(\mathcal{P}_\mathcal{L}) = \{[\![\shortparallel]\!]\}$.

Proof. First, we prove statement (1). The inclusion "$\subseteq$" is trivial since

$$\mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![\overline{r}w]\!] = \mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![\overline{r}]\!] \cdot [\![w]\!] \subseteq \mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![w]\!] .$$

For the converse inclusion let $[\![t]\!] \in \mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![w]\!]$. Then there is an $s \in \Sigma_{\shortparallel}^*$ with $t \equiv sw$. Set $s' := s \cdot r^{\mathrm{R}} \in \Sigma_{\shortparallel}^*$. Then we have

$$s' \cdot \overline{r}w = s \cdot r^{\mathrm{R}} \cdot \overline{r}w \equiv sw \equiv t$$

implying $[\![t]\!] \in \mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![\overline{r}w]\!]$.

The proof of the second statement is very similar to the first one. Next, we prove (3). This equation holds since:

$$\begin{aligned}
\mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![\overline{r}w]\!] \cdot \mathbb{T}(\mathcal{P}_\mathcal{L}) &= (\mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![\overline{r}w]\!]) \cdot \mathbb{T}(\mathcal{P}_\mathcal{L}) \overset{(1)}{=} (\mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot [\![w]\!]) \cdot \mathbb{T}(\mathcal{P}_\mathcal{L}) \\
&= \mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot ([\![w]\!] \cdot \mathbb{T}(\mathcal{P}_\mathcal{L})) \overset{(2)}{=} \mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot ([\![\varepsilon]\!] \cdot \mathbb{T}(\mathcal{P}_\mathcal{L})) \\
&= \mathbb{T}(\mathcal{P}_\mathcal{L}) \cdot \mathbb{T}(\mathcal{P}_\mathcal{L}) = \mathbb{T}(\mathcal{P}_\mathcal{L}) .
\end{aligned}$$

[vi] Let $\mathbb{M}$ be a monoid and $m \in \mathbb{M}$ be an element. Then $\mathbb{M} \cdot m$ is called a *left principal ideal* of $\mathbb{M}$. Similarly, $m \cdot \mathbb{M}$ is a *right principal ideal* and $\mathbb{M} \cdot m \cdot \mathbb{M}$ is a *two-sided principal ideal*.

Finally, statement (4) is a direct consequence of Lemma 4.4.1(4). ◂

With the help of this lemma, we are able to fully describe Green's relations in the partially lossy stack monoid:

Proposition 4.4.5. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $r_1, r_2, w_1, w_2 \in A^*$. Then the following statements hold:*

(1) $[\![\overline{r_1} w_1]\!] \mathrel{L} [\![\overline{r_2} w_2]\!]$ *if, and only if,* $w_1 = w_2$.

(2) $[\![\overline{r_1} w_1]\!] \mathrel{R} [\![\overline{r_2} w_2]\!]$ *if, and only if,* $r_1 = r_2$.

(3) $[\![\overline{r_1} w_1]\!] \mathrel{J} [\![\overline{r_2} w_2]\!]$.

(4) $[\![\overline{r_1} w_1]\!] \mathrel{H} [\![\overline{r_2} w_2]\!]$ *if, and only if,* $\overline{r_1} w_1 = \overline{r_2} w_2$.

(5) $[\![\overline{r_1} w_1]\!] \mathrel{D} [\![\overline{r_2} w_2]\!]$.

Now, let $t \in \Sigma_{\shortparallel}^*$. Then we have:

(6) $[\![\shortparallel]\!] \mathrel{X} [\![t]\!]$ *if, and only if,* $t \equiv \shortparallel$ *for each* $X \in \{L, R, J, H, D\}$.

Proof. We start with the proof of (1). If $w_1 = w_2$ holds, then we have, by Lemma 4.4.4(1),

$$\mathbb{T}(\mathcal{P}_{\mathcal{L}}) \cdot [\![\overline{r_1} w_1]\!] = \mathbb{T}(\mathcal{P}_{\mathcal{L}}) \cdot [\![w_1]\!] = \mathbb{T}(\mathcal{P}_{\mathcal{L}}) \cdot [\![w_2]\!] = \mathbb{T}(\mathcal{P}_{\mathcal{L}}) \cdot [\![\overline{r_2} w_2]\!]$$

implying $[\![\overline{r_1} w_1]\!] \mathrel{L} [\![\overline{r_2} w_2]\!]$. Conversely, if $[\![\overline{r_1} w_1]\!] \mathrel{L} [\![\overline{r_2} w_2]\!]$, we have

$$\mathbb{T}(\mathcal{P}_{\mathcal{L}}) \cdot [\![w_1]\!] = \mathbb{T}(\mathcal{P}_{\mathcal{L}}) \cdot [\![\overline{r_1} w_1]\!] = \mathbb{T}(\mathcal{P}_{\mathcal{L}}) \cdot [\![\overline{r_2} w_2]\!] = \mathbb{T}(\mathcal{P}_{\mathcal{L}}) \cdot [\![w_2]\!]$$

by Lemma 4.4.4(1). Since $[\![w_1]\!] \in \mathbb{T}(\mathcal{P}_{\mathcal{L}}) \cdot [\![w_1]\!] = \mathbb{T}(\mathcal{P}_{\mathcal{L}}) \cdot [\![w_2]\!]$ there is a $t \in \Sigma_{\shortparallel}^*$ with $w_1 \equiv t w_2$. Due to Theorem 4.4.2 we have

$$w_1 = \mathrm{nf}(w_1) = \mathrm{nf}(t w_2) = \mathrm{nf}(t) w_2$$

implying $\mathrm{nf}(t) \in A^*$ and w_2 is a suffix of w_1. By symmetry w_1 also is a suffix of w_2 implying $w_1 = w_2$.

Statement (2) can be proven similarly using Lemma 4.4.4(2).

Statement (3) holds since

$$\mathbb{T}(\mathcal{P}_{\mathcal{L}}) \cdot [\![\overline{r_1} w_1]\!] \cdot \mathbb{T}(\mathcal{P}_{\mathcal{L}}) = \mathbb{T}(\mathcal{P}_{\mathcal{L}}) = \mathbb{T}(\mathcal{P}_{\mathcal{L}}) \cdot [\![\overline{r_2} w_2]\!] \cdot \mathbb{T}(\mathcal{P}_{\mathcal{L}})$$

by Lemma 4.4.4(3).

The implication "$\Leftarrow$" of statement (4) is obvious. Towards the converse implication "$\Rightarrow$" let $[\![\overline{r_1} w_1]\!] \mathrel{H} [\![\overline{r_2} w_2]\!]$. Then we have $[\![\overline{r_1} w_1]\!] \mathrel{L} [\![\overline{r_2} w_2]\!]$ and $[\![\overline{r_1} w_1]\!] \mathrel{R} [\![\overline{r_2} w_2]\!]$. By statements (1) and (2) we have $w_1 = w_2$ and $r_1 = r_2$ implying $\overline{r_1} w_1 = \overline{r_2} w_2$.

Next we prove (5). Set $t := \overline{r_2} w_1$. Then by the choice of t we have $[\![\overline{r_1} w_1]\!] \mathrel{L} [\![t]\!]$ and $[\![t]\!] \mathrel{R} [\![\overline{r_2} w_2]\!]$ implying $[\![\overline{r_1} w_1]\!] \mathrel{D} [\![\overline{r_2} w_2]\!]$.

Finally, we can infer (6) directly from Lemma 4.4.4(4). ◂

4.4.4 Composition

Finally, we characterize the normal form of the composition of two transformations. We first specify the normal form of a special case, namely of action sequences from $A^*\overline{A}^*$. To this end, we first have to understand, in which cases an action sequence $w\overline{a}$ (for $w \in A^*$ and $a \in A$) behaves equivalently to ε. In reliable stacks we have $w\overline{a} \equiv \varepsilon$ if, and only if, $w = a$ holds due to Lemma 4.3.1(1). In non-reliable stacks we additionally have equation (2) from Lemma 4.4.1. This means, before we apply the equation $a\overline{a} \equiv \varepsilon$ to $w\overline{a}$ we can also remove some forgettable letters other than a. Then we obtain $w\overline{a} \equiv \varepsilon$ if, and only if, $w \in a(F \setminus \{a\})^*$ holds. Now, let $r \in A^*$. Then by induction on the length of r we obtain $w\overline{r} \equiv \varepsilon$ if, and only if, $r = a_1a_2 \dots a_n$ and $w \in a_n(F \setminus \{a_n\})^* \dots a_2(F \setminus \{a_2\})^* a_1(F \setminus \{a_1\})^*$. In this case, w^{R} is an $\mathcal{L}$-superword of r with some restrictions:

Definition 4.4.6. Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $u, v \in A^*$. We say that u is a *reduced $\mathcal{L}$-superword* of v if there are letters $a_1, a_2, \dots, a_n \in A$ and words $x_i \in (F \setminus \{a_i\})^*$ such that $v = a_1a_2 \dots a_n$ and $u = x_1a_1x_2a_2 \dots x_na_n$ holds.

The set of all reduced $\mathcal{L}$-superwords of $v = a_1a_2 \dots a_n$ is denoted by

$$\mathrm{redsup}_{\mathcal{L}}(v) := \{x_1a_1x_2a_2 \dots x_na_n \mid \forall 1 \le i \le n\colon x_i \in (F \setminus \{a_i\})^*\}.$$

Then we see that $w\overline{r} \equiv \varepsilon$ holds if, and only if, $w^{\mathrm{R}} \in \mathrm{redsup}_{\mathcal{L}}(r)$ for each pair $r, w \in A^*$. Note that the relation "$u \in \mathrm{redsup}_{\mathcal{L}}(v)$" is (in the general case) not transitive, i.e., this relation is no partial order. Though, in the reliable case (i.e., $\mathcal{L} = (\varnothing, U)$) we have $\mathrm{redsup}_{\mathcal{L}}(v) = \{v\}$. In other words, the mentioned relation is the equality relation on the free monoid A^*.

All in all, we obtain the following three cases concerning the normal form of $w\overline{r}$ for arbitrary $r, w \in A^*$:

Lemma 4.4.7. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $w, r \in A^*$. Let $r_1 \in A^*$ be the longest prefix of r such that there exists a suffix $w_2 \in A^*$ of w with $w_2{}^{\mathrm{R}} \in \mathrm{redsup}_{\mathcal{L}}(r_1)$. Moreover, let r_2 be the complementary suffix of r wrt. r_1 and let w_1 be the complementary prefix of w wrt. w_2. Then the following statements hold:*

(1) if $r_2 = \varepsilon$ then we have $\mathrm{nf}(w\overline{r}) = w_1$.

(2) if $r_2 \neq \varepsilon$ and $w_1 \in F^*$ then we have $\mathrm{nf}(w\overline{r}) = \overline{r_2}$.

(3) if $r_2 \neq \varepsilon$ and $w_1 \notin F^*$ then we have $\mathrm{nf}(w\overline{r}) = \bot\!\!\!\bot$.

Proof. There are letters $a_1, \dots, a_n \in A$ and $x_i \in (F \setminus \{a_i\})^*$ such that $r_1 = a_1 \dots a_n$ and $w_2 = a_nx_n \dots a_1x_1$. Then we can see the following:

$$\begin{aligned}
w\overline{r} &= w_1\, w_2\, \overline{r_1}\, \overline{r_2} = w_1\, a_nx_n \dots a_2x_2a_1x_1\, \overline{a_1a_2} \dots \overline{a_n}\, \overline{r_2} \\
&\equiv w_1\, a_nx_n \dots a_2x_2a_1\, \overline{a_1a_2} \dots \overline{a_n}\, \overline{r_2} && \text{(by Lemma 4.4.1(2))} \\
&\equiv w_1\, a_nx_n \dots a_2x_2\, \overline{a_2} \dots \overline{a_n}\, \overline{r_2} && \text{(by Lemma 4.4.1(1))} \\
&\;\;\vdots \\
&\equiv w_1\, \overline{r_2}\,.
\end{aligned}$$

Then statement (1) is a simple consequence of these calculations.

Towards (2) let $r_2 = a_{n+1}r_2'$ with $a_{n+1} \in A$. Due to the maximality of $|r_1|$ we have $w_1 \in (F \setminus \{a_{n+1}\})^*$. Hence, using Lemma 4.4.1(2) we obtain $w\overline{r} \equiv w_1\overline{r_2} \equiv \overline{r_2}$.

Finally, we prove (3). By the maximality of $|r_1|$ we obtain $w_2 \notin A^* a_{n+1}(F \setminus \{a_{n+1}\})^*$. Then Lemma 4.4.1(3) yields $w\overline{r} \equiv \perp\!\!\!\perp$. ◂

From this lemma we obtain our characterization of the composition of two arbitrary action sequences:

Theorem 4.4.8. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $s, t \in \Sigma_{\perp\!\!\!\perp}^*$ with $\mathrm{nf}(s) = \overline{r_1}w_1$ and $\mathrm{nf}(t) = \overline{r_2}w_2$ for words $w_1, w_2, r_1, r_2 \in A^*$. Set $v := \mathrm{nf}(w_1\overline{r_2})$ (by Lemma 4.4.7 we know $v \in A^* \cup \overline{A}^* \cup \{\perp\!\!\!\perp\}$). If $v \neq \perp\!\!\!\perp$ then we have $\mathrm{nf}(st) = \overline{r_1}vw_2$. Otherwise we have $\mathrm{nf}(st) = \perp\!\!\!\perp$.* ◂

4.5 Distributed Stacks

In this section we want to generalize the results from Section 4.3 to systems having multiple stacks. We first show that the behavioral equivalence is compatible with the partial commutations defined by the underlying distributed alphabet $\mathcal{A}$. So, we can also understand our action sequences as traces. Afterwards, we will consider a trace rewriting system[vii] from which we obtain a normal form of the equivalence classes.

Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet. For a letter $a \in A$ the transformation $[\![\overline{a}]\!]$ is the right-inverse element of $[\![a]\!]$ but not its left-inverse. Now, let $a, b \in A$ be two distinct letters. If $a \parallel b$ (i.e., a and b affect different stacks) then actions $\alpha \in \{a, \overline{a}\}$ and $\beta \in \{b, \overline{b}\}$ commute. This does not hold if $a \nparallel b$ holds. In particular, we observe

$$[\![a\overline{b}]\!](\lambda) = [\![\overline{b}]\!](a\lambda) = \bot$$

in this case for any trace $\lambda \in \mathbb{M}(\mathcal{A})$. Hence, the transformation $[\![a\overline{b}]\!]$ is an annihilating element of $\mathbb{T}(\mathcal{P}_{\mathcal{A}})$. We introduce a new letter $\perp\!\!\!\perp \notin \Sigma$ with $[\![\perp\!\!\!\perp]\!](\lambda) = \bot$ for each $\lambda \in \mathbb{M}(\mathcal{A}) \cup \{\bot\}$. The extended set of basic actions is then $\Sigma_{\perp\!\!\!\perp} := \Sigma \cup \{\perp\!\!\!\perp\}$. Note that there is no such annihilating element $[\![\perp\!\!\!\perp]\!]$ in $\mathbb{T}(\mathcal{P}_{\mathcal{A}})$ if we have $a \parallel b$ for each pair of distinct letters $a, b \in A$ (in this case, $\mathcal{P}_{\mathcal{A}}$ consists of multiple independent counters).

First, we generalize the equations from Lemma 4.3.1 to the distributed case. These equations are listed in the next lemma:

Lemma 4.5.1. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $a, b \in A$, and $\alpha \in \Sigma_{\perp\!\!\!\perp}$. Then the following equations hold:*

(1) $a\overline{a} \equiv \varepsilon$,

[vii] *Trace rewriting systems* are a generalization of semi-Thue systems which handle traces instead of words. Properties like termination or confluence are similarly defined. More information on such systems can be found, e.g., in [Die90].

(2) $ab \equiv ba$, $\overline{a}\overline{b} \equiv \overline{b}\overline{a}$, *and* $a\overline{b} \equiv \overline{b}a$ *if* $a \parallel b$,

(3) $a\overline{b} \equiv \perp\!\!\!\perp$ *if* $a \neq b$ *and* $a \nparallel b$, *and*

(4) $\alpha \perp\!\!\!\perp \equiv \perp\!\!\!\perp \alpha \equiv \perp\!\!\!\perp$.

Proof. We start with the proof of statement (1). To this end, let $a \in A$ and $\lambda \in \mathbb{M}(\mathcal{A})$. Then we see

$$[\![a\overline{a}]\!](\lambda) = [\![\overline{a}]\!](a\lambda) = \lambda = [\![\varepsilon]\!](\lambda)\,.$$

This implies $[\![a\overline{a}]\!] = [\![\varepsilon]\!]$ and, therefore, $a\overline{a} \equiv \varepsilon$.

Towards statement (2) assume that $a \parallel b$ holds. First we prove $ab \equiv ba$. Then we have for $\lambda \in \mathbb{M}(\mathcal{A})$:

$$[\![ab]\!](\lambda) = [\![b]\!]([\![a]\!](\lambda)) = ba\lambda = ab\lambda = [\![a]\!]([\![b]\!](\lambda)) = [\![ba]\!](\lambda)\,.$$

Next, we consider $\overline{a}\overline{b} \equiv \overline{b}\overline{a}$. Let $\lambda \in \mathbb{M}(\mathcal{A})$ with $\kappa := [\![\overline{a}\overline{b}]\!](\lambda) \neq \bot$. Then we obtain $\lambda = ab\kappa = ba\kappa$ by $a \parallel b$. This implies

$$[\![\overline{a}\overline{b}]\!](\lambda) = \kappa = [\![\overline{a}]\!](a\kappa) = [\![\overline{b}\overline{a}]\!](ba\kappa) = [\![\overline{b}\overline{a}]\!](\lambda)\,.$$

Conversely, let $\lambda \in \mathbb{M}(\mathcal{A})$ with $[\![\overline{b}\overline{a}]\!](\lambda) \neq \bot$. Then by symmetry we infer $[\![\overline{b}\overline{a}]\!](\lambda) = [\![\overline{a}\overline{b}]\!](\lambda)$.

Now, we prove the third equation $a\overline{b} \equiv \overline{b}a$. First, let $\lambda \in \mathbb{M}(\mathcal{A})$ be a distributed stack's content with $\kappa := [\![a\overline{b}]\!](\lambda) \neq \bot$. Then we see $a\lambda = b\kappa$. Since $a \parallel b$ holds, there is $\mu \in \mathbb{M}(\mathcal{A})$ with $b\kappa = a\lambda = ab\mu = ba\mu$ and, hence, $\kappa = a\mu$ and $\lambda = b\mu$ by the cancellation property of $\mathbb{M}(\mathcal{A})$. Then we have

$$[\![a\overline{b}]\!](\lambda) = \kappa = a\mu = [\![a]\!](\mu) = [\![\overline{b}a]\!](b\mu) = [\![\overline{b}a]\!](\lambda)\,.$$

Conversely, let $\lambda \in \mathbb{M}(\mathcal{A})$ with $\kappa := [\![\overline{b}a]\!](\lambda) \neq \bot$. We obtain a trace $\mu \in \mathbb{M}(\mathcal{A})$ with $\lambda = b\mu$ and $\kappa = a\mu$. Then we have

$$[\![\overline{b}a]\!](\lambda) = \kappa = a\mu = [\![\overline{b}]\!](ba\mu) = [\![\overline{b}]\!](ab\mu) = [\![a\overline{b}]\!](b\mu) = [\![a\overline{b}]\!](\lambda)\,.$$

Towards statement (3), let $a, b \in A$ be two distinct letters with $a \nparallel b$ and let $\lambda \in \mathbb{M}(\mathcal{A})$. By the choice of a and b we know $ab \not\sim_{\mathcal{A}} ba$ and, hence, $a\lambda \neq b\kappa$ for each $\kappa \in \mathbb{M}(\mathcal{A})$. Then we obtain

$$[\![a\overline{b}]\!](\lambda) = [\![\overline{b}]\!](a\lambda) = \bot = [\![\perp\!\!\!\perp]\!](\lambda)$$

implying $a\overline{b} \equiv \perp\!\!\!\perp$.

Finally, we consider the last statement. So, let $s, t \in \Sigma_{\perp\!\!\!\perp}^*$ and $\lambda \in \mathbb{M}(\mathcal{A})$. Then we have

$$[\![s \perp\!\!\!\perp t]\!](\lambda) = [\![t]\!]([\![\perp\!\!\!\perp]\!]([\![s]\!](\lambda))) = [\![t]\!](\bot) = \bot = [\![\perp\!\!\!\perp]\!](\lambda)\,,$$

which implies $s \perp\!\!\!\perp t \equiv \perp\!\!\!\perp$. Since s and t are arbitrary action sequences, this equation also holds for $st = \alpha$ with $\alpha \in \Sigma_{\perp\!\!\!\perp}$. ◂

Consider the equations from Lemma 4.4.1(3). Then we can see that write and read actions of two independent letters $a \parallel b$ are commutable in the behavioral equivalence. In other words, the behavioral equivalence preserves the partial commutations defined by an extension of the distributed alphabet $\mathcal{A}$ to all basic actions of a distributed stack. This extended distributed alphabet is $\mathcal{E}_{\perp\!\!\!\perp} := (\Sigma_{\perp\!\!\!\perp}, P, \mathrm{M}')$ with $\mathrm{M}' := \mathrm{M} \cup \{(\overline{a}, i) \mid (a, i) \in \mathrm{M}\} \cup (\{\perp\!\!\!\perp\} \times P)$. Finally, we observe that $\equiv$ is a quotient of the partial commutations $\approx_{\mathcal{E}_{\perp\!\!\!\perp}}$. For a trace $\tau \in \mathbb{M}(\mathcal{A})$ we denote $\overline{\tau} := [\overline{w}]$ for a word $w \in \tau$. Additionally, for a trace language $L \subseteq \mathbb{M}(\mathcal{A})$ we write $\overline{L} := \{\overline{\tau} \mid \tau \in L\}$.

From now on we can also define the distributed stack's semantics $[\![.]\!]$ for traces. So, we write $[\![\tau]\!]$ for the transformation $[\![t]\!]$ with $t \in \tau$ for each $\tau \in \mathbb{M}(\mathcal{E}_{\perp\!\!\!\perp})$. This definition is well-defined due to the argumentation above. Moreover, we extend our behavioral equivalence to traces $\sigma, \tau \in \mathbb{M}(\mathcal{E}_{\perp\!\!\!\perp})$. Then we write $\sigma \equiv \tau$ if, and only if, $[\![\sigma]\!] = [\![\tau]\!]$ holds. This is the case whenever $s \equiv t$ holds for $s \in \sigma$ and $t \in \tau$. Then with trace semantics Lemma 4.5.1 reduces to the equations (1), (3), and (4). Note that these are essentially the equations from Lemma 4.3.1.

Now, we order these equations from left to right and obtain a trace rewriting system $\mathfrak{S}_{\mathcal{A}}$ (on the distributed alphabet $\mathcal{E}_{\perp\!\!\!\perp}$) with the following rules:

(1) $[a\overline{a}] \to [\varepsilon]$,

(2) $[a\overline{b}] \to [\perp\!\!\!\perp]$ if $a \neq b$ and $a \nparallel b$,

(3) $[\alpha \perp\!\!\!\perp] \to [\perp\!\!\!\perp]$, and $[\perp\!\!\!\perp \alpha] \to [\perp\!\!\!\perp]$

where $a, b \in A$ are two letters and $\alpha \in \Sigma_{\perp\!\!\!\perp}$. Note that this trace rewriting system is finite and monadic. It is also terminating since $\sigma \Rightarrow_{\mathfrak{S}_{\mathcal{A}}} \tau$ implies $|\sigma| > |\tau|$ for each pair of traces $\sigma, \tau \in \mathbb{M}(\mathcal{E}_{\perp\!\!\!\perp})$. Additionally, we can see that the system is confluent: to this end, we only have to consider overlapping left-hand sides of the rules in $\mathfrak{S}_{\mathcal{A}}$. This is the case for $[a\overline{b}\perp\!\!\!\perp]$ and $[\perp\!\!\!\perp a\overline{b}]$ (where a and b are not necessarily distinct). Due to symmetry we only consider the former case:

- $[\perp\!\!\!\perp] \overset{(3)}{\Longleftarrow}_{\mathfrak{S}_{\mathcal{A}}} [a\perp\!\!\!\perp] \overset{(3)}{\Longleftarrow}_{\mathfrak{S}_{\mathcal{A}}} [a(\overline{a}\perp\!\!\!\perp)] = [(a\overline{a})\perp\!\!\!\perp] \overset{(1)}{\Longrightarrow} \mathfrak{S}_{\mathcal{A}}[\perp\!\!\!\perp]$ or
- $[\perp\!\!\!\perp] \overset{(3)}{\Longleftarrow}_{\mathfrak{S}_{\mathcal{A}}} [a\perp\!\!\!\perp] \overset{(3)}{\Longleftarrow}_{\mathfrak{S}_{\mathcal{A}}} [a(\overline{b}\perp\!\!\!\perp)] = [(a\overline{b})\perp\!\!\!\perp] \overset{(2)}{\Longrightarrow}_{\mathfrak{S}_{\mathcal{A}}} [\perp\!\!\!\perp\perp\!\!\!\perp] \overset{(3)}{\Longrightarrow}_{\mathfrak{S}_{\mathcal{A}}} [\perp\!\!\!\perp]$ if $a \neq b$ and $a \nparallel b$.

Since $\mathfrak{S}_{\mathcal{A}}$ is terminating and confluent, for each trace $\tau \in \mathbb{M}(\mathcal{E}_{\perp\!\!\!\perp})$ there is a unique, irreducible trace which we call $\mathrm{nf}(\tau)$ (the so-called *normal form*) and which satisfies $\tau \Rightarrow^*_{\mathfrak{S}_{\mathcal{A}}} \mathrm{nf}(\tau)$. The set of all traces in normal form is denoted by $\mathrm{NF}_{\mathcal{P}_{\mathcal{A}}}$. Again, we can obtain the following characterization of the normal forms from the rules of $\mathfrak{S}_{\mathcal{A}}$:

$$\mathrm{NF}_{\mathcal{P}_{\mathcal{A}}} = \overline{\mathbb{M}(\mathcal{A})}\mathbb{M}(\mathcal{A}) \cup \{[\perp\!\!\!\perp]\}\,. \tag{4.5}$$

We prove next, that the normal form is unique in any equivalence class of $\equiv$. This implies that the equations from Lemma 4.5.1 fully describe the behavioral equivalence:

Theorem 4.5.2. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\sigma, \tau \in \mathbb{M}(\mathcal{E}_{\perp\!\!\!\perp})$. Then we have*

$$\sigma \equiv \tau \iff \mathrm{nf}(\sigma) = \mathrm{nf}(\tau)$$

Proof. The proof of this theorem is very similar to the proof of Theorem 4.4.2. ◂

This finally implies that the map $\mathbb{T}(\mathcal{P}_{\mathcal{A}}) \to \mathrm{NF}_{\mathcal{P}_{\mathcal{A}}} \colon [\![\tau]\!] \mapsto \mathrm{nf}(\tau)$ is well-defined. We can also write $\mathrm{nf}([\![\tau]\!])$ for the normal form $\mathrm{nf}(\tau)$ of the trace of actions $\tau \in \mathbb{M}(\mathcal{E}_{\shuffle})$. Note that due to Equation (4.5) and Theorem 4.5.2 for each trace $\tau \in \mathbb{M}(\mathcal{E}_{\shuffle})$ there is an equivalently behaving trace $\sigma \in \mathbb{M}(\mathcal{E}_{\shuffle})$ which is in some sense simple.

4.6 Recapitulation: Reliable Queues

In this section we want to recapitulate the results from [HKZ17] concerning the basic properties of the (reliable) queue monoid $\mathbb{T}(\mathcal{Q}_A)$. The concrete structure of this section is similar to the previous ones. So, we start with listing several equations that hold in the behavioral equivalence of a reliable queue. From these equations we construct a semi-Thue system $\mathfrak{R}_A$. We will see that each equivalence class has a unique, irreducible word - the so-called normal form. So, we will infer that the equations fully describe the behavior of a reliable queue. Finally, we describe the composition of two transformations in terms of normal forms.

Before we do this, we first consider a special dualism of write and read actions. To this end, let $a \in A$ and $x \in A^*$. Then we can observe that $[\![a]\!](x) = xa$ and $[\![\overline{a}]\!](a\,x^{\mathrm{R}}) = x^{\mathrm{R}}$ holds. So, we can see that $[\![\overline{a}]\!]$ behaves as the inverse function of $[\![a]\!]$ if we reverse the direction of our queue. This duality can be extended to the so-called *duality map* $\mathrm{d} \colon \Sigma^* \to \Sigma^*$ which is defined as follows:

$$\mathrm{d}(\varepsilon) = \varepsilon\,, \qquad \mathrm{d}(at) = \mathrm{d}(t)\overline{a}\,, \text{ and} \qquad \mathrm{d}(\overline{a}t) = \mathrm{d}(t)a$$

for each $a \in A$ and $t \in \Sigma^*$. Note that d is a bijective antimorphism (i.e., we have $\mathrm{d}(st) = \mathrm{d}(t)\,\mathrm{d}(s)$) and an involution (i.e., we have $\mathrm{d}(\mathrm{d}(t)) = t$ for each $t \in \Sigma^*$). Then we can show that the behavioral equivalence is compatible with the duality map:

Proposition 4.6.1 ([HKZ17, Lemma 3.3 and Proposition 3.4]). *Let A be an alphabet. Then the following statements hold:*

(1) Let $t \in \Sigma^*$ and $x, y \in A^*$ with $[\![t]\!](x) = y \neq \bot$. Then we have $[\![\mathrm{d}(t)]\!](y^{\mathrm{R}}) = x^{\mathrm{R}}$.

(2) Let $s, t \in \Sigma^*$. Then we have $s \equiv t$ if, and only if, $\mathrm{d}(s) \equiv \mathrm{d}(t)$ holds. ◂

Note that we will use this duality map and its properties multiple times in this thesis.

4.6.1 Normal Forms

Next, we want to list a finite number of (context-sensitive) commutations that hold in the behavioral equivalence. Later we will state that these equations fully describe the equivalence classes of $\equiv$.

Lemma 4.6.2 ([HKZ17, Lemma 3.5]). *Let A be an alphabet and $a, b \in A$. Then the following equations hold:*

(1) $a\overline{b} \equiv \overline{b}a$ *if* $a \neq b$,

(2) $a\overline{a}\overline{b} \equiv \overline{a}a\overline{b}$, *and*

(3) $ba\overline{a} \equiv b\overline{a}a$. ◂

We can understand the first equation as follows: in order for $[\![a\overline{b}]\!](x)$ to be defined, the queue's content x already has to start with the letter b. Then it does not matter whether we first write a and then read b or vice versa. We are in a similar situation in the case $[\![a\overline{a}\overline{b}]\!](x) \neq \bot$. This only holds if the queue's content starts with an a. The third equation can be obtained from the second one using the duality property (Proposition 4.6.1).

Now, we are able to construct a semi-Thue system $\mathfrak{R}_A$ by ordering the equations from Lemma 4.6.2 from left to right. In other words, $\mathfrak{R}_A$ consists of the following rules:

(1) $a\overline{b} \to \overline{b}a$ if $a \neq b$,

(2) $a\overline{a}\overline{b} \to \overline{a}a\overline{b}$, and

(3) $ba\overline{a} \to b\overline{a}a$

for each $a, b \in A$. We observe that each rule in $\mathfrak{R}_A$ commutes a write and a read action. So, if we apply these rules to an action sequence $t \in \Sigma^*$ we preserve the relative order of write actions and the relative order of read actions. We denote the subsequences of write and read actions of t by $\mathrm{wrt}(t) := \pi_A(t)$ and $\overline{\mathrm{rd}(t)} := \pi_{\overline{A}}(t)$ (note that rd suppresses the bars on top of the read actions in t).

The semi-Thue system $\mathfrak{R}_A$ is finite, terminating, and confluent [HKZ17, Lemma 4.1]. Then, for each action sequence $t \in \Sigma^*$ we obtain a unique, irreducible word $\mathrm{nf}(t)$ (the so-called *normal form*) with $t \Rightarrow_{\mathfrak{R}_A}^* \mathrm{nf}(t)$. The set of all action sequences in normal form is denoted by NF_{Q_A}. Due to the shape of the rules of $\mathfrak{R}_A$ we obtain

$$\mathrm{NF}_{Q_A} = \overline{A}^* \{a\overline{a} \mid a \in A\}^* A^* .$$

For letters $a_1, a_2, \ldots, a_n \in A$ we write $\langle a_1 a_2 \ldots a_n \rangle := a_1\overline{a_1}a_2\overline{a_2}\ldots a_n\overline{a_n} \in \Sigma^*$. Then, the normal form of an action sequence $t \in \Sigma^*$ is $\mathrm{nf}(t) = \overline{w_1}\langle w_2 \rangle w_3$ for three uniquely defined words $w_1, w_2, w_3 \in A^*$. We also name these words by $\mathrm{rd}_1(t) := w_1$, $\mathrm{rd}_2(t) = \mathrm{wrt}_1(t) := w_2$, and $\mathrm{wrt}_2(t) := w_3$. Note that we have $\mathrm{wrt}(t) = \mathrm{wrt}_1(t)\mathrm{wrt}_2(t)$ and $\mathrm{rd}(t) = \mathrm{rd}_1(t)\mathrm{rd}_2(t)$.

Finally, we can prove that the equations from Lemma 4.6.2 fully describe the behavioral equivalence. In other words, the normal forms are uniquely defined for equivalence classes of $\equiv$.

Theorem 4.6.3 ([HKZ17, Theorem 4.3]). *Let A be an alphabet and $s, t \in \Sigma^*$. Then we have $s \equiv t$ if, and only if,* $\mathrm{nf}(s) = \mathrm{nf}(t)$. ◂

So, we can also see the normal form $\mathrm{nf}(t)$ of an action sequence $t \in \Sigma^*$ as the normal form $\mathrm{nf}(\llbracket t \rrbracket)$ of the induced transformation $\llbracket t \rrbracket$. Similarly, we can define the maps rd, rd_i, wrt, and wrt_i (for $i = 1, 2$) for transformations from $\mathbb{T}(Q_A)$. Additionally, the normal form of t (resp. $\llbracket t \rrbracket$) is uniquely described by the words $\mathrm{wrt}(t)$, $\mathrm{rd}(t)$, and $\mathrm{rd}_2(t) = \mathrm{wrt}_1(t)$. We call the triple $\chi(t) := (\mathrm{wrt}(t), \mathrm{rd}(t), \mathrm{rd}_2(t))$ the *characteristic* of t (resp. $\llbracket t \rrbracket$).

4.6.2 Cancellation and Green's Relations

From the equations in Lemma 4.6.2 and the fact that these equations fully describe the queue monoid (cf. Theorem 4.6.3), we also obtain that $\mathbb{T}(Q_A)$ is not cancellative. For example, we know $\llbracket a\overline{a} \rrbracket \neq \llbracket \overline{a}a \rrbracket$ but $\llbracket a \rrbracket \llbracket a\overline{a} \rrbracket = \llbracket a \rrbracket \llbracket \overline{a}a \rrbracket$ and $\llbracket a\overline{a} \rrbracket \llbracket \overline{a} \rrbracket = \llbracket \overline{a}a \rrbracket \llbracket \overline{a} \rrbracket$ for any $a \in A$. However, the queue monoid is cancellative under the same restrictions as we have seen for the stack monoids (cf. Corollary 4.4.3). We also learn that Green's relations are trivial in the queue monoid, i.e., all of these relations coincide with the identity relation.

Corollary 4.6.4 ([HKZ17, Corollaries 4.6 and 4.7]). *Let A be an alphabet. Then the following statements hold:*

(1) $\mathbb{T}(Q_A)$ is neither left- nor right-cancellative.

(2) If $s, t \in \Sigma^*$ and $x, y \in A^*$ with $\overline{x}sy \equiv \overline{x}ty$ holds, then we have $s \equiv t$.

(3) For all $s, t \in \Sigma^*$ and $\mathrm{X} \in \{\mathrm{L}, \mathrm{R}, \mathrm{J}, \mathrm{H}, \mathrm{D}\}$ we have $\llbracket s \rrbracket \,\mathrm{X}\, \llbracket t \rrbracket$ if, and only if, $s \equiv t$. ◂

4.6.3 Composition

Finally, we can study the normal form of the composition of two action sequences. But before we consider the general case, we consider a crucial special case. Concretely, we compute the normal form of the action sequence $w\overline{r}$ where $w, r \in A^*$. We consider the characteristic $\chi(w\overline{r}) = (\mathrm{wrt}(w\overline{r}), \mathrm{rd}(w\overline{r}), \mathrm{rd}_2(w\overline{r}))$ (recall that $\chi(w\overline{r})$ fully describes the normal form of $w\overline{r}$). By the definition of the projections to write and read actions, we have $\mathrm{wrt}(w\overline{r}) = w$ and $\mathrm{rd}(w\overline{r}) = r$. So, we only have to determine $\mathrm{rd}_2(w\overline{r})$. First, from $\mathrm{nf}(w\overline{r}) = \overline{\mathrm{rd}_1(w\overline{r})}\langle \mathrm{rd}_2(w\overline{r}) \rangle \mathrm{wrt}_2(w\overline{r})$ we can learn that $\mathrm{rd}_2(w\overline{r}) = \mathrm{wrt}_1(w\overline{r})$ is a suffix of $\mathrm{rd}(w\overline{r}) = r$ and a prefix of $\mathrm{wrt}(w\overline{r}) = w$.

Example 4.6.5. Let $a, b \in A$ be two disjoint letters. We consider the words $w = r := ababa$. Then there are the following four suffixes of r which also are prefixes of w: ε, a, aba, and $ababa$. However, the application of the equations from Lemma 4.6.2 yields the following:

$$ababa\overline{ababa} \equiv a\overline{a}b\overline{b}a\overline{a}b\overline{b}a\overline{a} = \langle ababa \rangle \in \mathrm{NF}_{Q_A}$$

implying $\mathrm{rd}_2(ababa\overline{ababa}) = ababa$. In other words, $\mathrm{rd}_2(ababa\overline{ababa})$ is the longest of the four aforementioned candidates. ⌟

Finally, Huschenbett et al. proved that $\mathrm{rd}_2(w\overline{r})$ always has maximal length:

Lemma 4.6.6 ([HKZ17, Lemma 5.2]). *Let A be an alphabet and $w, r \in A^*$. Set y to the longest suffix of r which also is a prefix of w. Then we have $\mathrm{rd}_2(w\overline{r}) = y$. In particular, for $x, z \in A^*$ with $r = xy$ and $w = yz$ we have $\mathrm{nf}(w\overline{r}) = \overline{x}\langle y\rangle z$.* ◂

From this result we also learn that each action sequence behaves equivalently to a somewhat simple action sequence, namely a sequence from $\overline{A}^* A^* \overline{A}^*$:

Corollary 4.6.7. *Let A be an alphabet and $t \in \Sigma^*$ be an action sequence. Then we have $t \equiv \overline{\mathrm{rd}_1(t)}\mathrm{wrt}(t)\overline{\mathrm{rd}_2(t)}$.* ◂

Note that there are action sequences $t \in \Sigma^*$ having multiple simple action sequences which behave equivalently. To this end, let $a, b \in A$ be two distinct letters. Then we have $ba\overline{a}\overline{b}\overline{a} \equiv \overline{a}ba\overline{b}\overline{a}$ by Lemma 4.6.6. Now, consider the words $r_1, r_2, w \in A^*$ where r_2 has minimal length with $t \equiv \overline{r_1} w \overline{r_2}$. We can see that $\overline{r_1} w \overline{r_2} = \overline{\mathrm{rd}_1(t)}\mathrm{wrt}(t)\overline{\mathrm{rd}_2(t)}$.

Using this corollary, we can also compute such simple equivalently behaving action sequence from a given action sequence $t \in \Sigma^*$ (in polynomial time): first we compute $\mathrm{nf}(t)$ using the semi-Thue system $\mathfrak{R}_A$. In this system, each derivation starting with t has length $O(|t|^2)$ since each step commutes a read and write action of t. From $\mathrm{nf}(t)$ we can finally extract $\mathrm{rd}_1(t)$, $\mathrm{wrt}(t)$, and $\mathrm{rd}_2(t)$.

Finally, we obtain the following general statement about the normal form of st where $s, t \in \Sigma^*$ are arbitrary action sequences:

Theorem 4.6.8 ([HKZ17, Theorem 5.3]). *Let A be an alphabet and $s, t \in \Sigma^*$. Set y to the longest suffix of $\mathrm{rd}_2(s)\mathrm{rd}(t)$ which also is a prefix of $\mathrm{wrt}(s)\mathrm{rd}_2(t)$. Then we have $\mathrm{rd}_2(st) = y$. In particular, for $x, z \in A^*$ with $\mathrm{rd}_2(s)\mathrm{rd}(t) = xy$ and $\mathrm{wrt}(s)\mathrm{rd}_2(t) = yz$ we have $\mathrm{nf}(st) = \overline{x}\langle y\rangle z$.* ◂

4.7 Partially Lossy Queues

Now, we want to generalize the results from the previous section to the behavioral equivalence of the partially lossy queue monoid. Again, we will first define a semi-Thue system on Σ^* which identifies equivalently behaving action sequences. With the help of this system we obtain special normal forms of the equivalence classes. Additionally, we will characterize the composition of two action sequences, cancellation properties, and Green's relations. In this connection we also recall the embedding properties into the plq monoid. Note that most of the proofs and results of this section stem from [Köc16, KKP18].

The following example shows that most of the results in this section heavily depend on the underlying lossiness alphabet of the partially lossy queue:

Example 4.7.1. Let A be an alphabet and $a, b \in A$ be two distinct letters. Then we have

$$[\![ba\overline{a}]\!]_{\mathcal{Q}_{(A,\varnothing)}}(\varepsilon) = [\![\overline{a}]\!]_{\mathcal{Q}_{(A,\varnothing)}}(ba) = \varepsilon \neq \bot = [\![\overline{a}a]\!]_{\mathcal{Q}_{(A,\varnothing)}}(b) = [\![b\overline{a}a]\!]_{\mathcal{Q}_{(A,\varnothing)}}(\varepsilon)$$

implying $ba\overline{a} \not\equiv_{\mathcal{Q}_{(A,\varnothing)}} b\overline{a}a$. In contrast, we have $[\![.]\!]_{\mathcal{Q}_{(\varnothing,A)}} = [\![.]\!]_{\mathcal{Q}_A}$ implying $ba\overline{a} \equiv_{\mathcal{Q}_{(\varnothing,A)}} b\overline{a}a$ by Lemma 4.6.2. ⌟

Note that for a singleton $A = \{a\}$ we have $\mathcal{Q}_{(A,\varnothing)} = \mathcal{Q}_{(\varnothing,A)}$. Moreover, these partially lossy queues are essentially counters as the following two equations prove:

$$[\![a]\!](a^n) = a^n a = a^{n+1} \qquad \text{and} \qquad [\![\overline{a}]\!](a^{n+1}) = [\![\overline{a}]\!](aa^n) = a^n$$

for each $n \in \mathbb{N}$. In other words, $\mathbb{T}(\mathcal{Q}_{(A,\varnothing)}) = \mathbb{T}(\mathcal{Q}_{(\varnothing,A)}) \cong \mathbb{T}(C_1) \cong \mathbb{T}(\mathcal{P}_A)$ is the bicyclic semigroup. Some of the results in this section do not hold in these cases. However, we have already seen some properties of the behavioral equivalence of these special plqs in Section 4.3.

4.7.1 Basic Properties

The following lemma lists some context-sensitive commutations that hold in the behavioral equivalence. Later in this section (in Theorem 4.7.8) we will see that these equations fully describe the behavioral equivalence.

Lemma 4.7.2. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $a, b \in A$, and $t \in A^*$. Then the following equations hold:*

(1) $b\overline{a} \equiv \overline{a}b$ *if* $a \neq b$,

(2) $a\overline{ab} \equiv \overline{a}a\overline{b}$,

(3) $bta\overline{a} \equiv bt\overline{a}a$ *if* $b \in U$, *and*

(4) $ata\overline{a} \equiv at\overline{a}a$.

Before we prove this lemma, we want to take a look on these equations. We first consider the equation (1). In order for a partially lossy queue with content $x \in A^*$ to be defined after application of the transformation $[\![\overline{a}b]\!]$, x already has to contain the letter a preceded by forgettable letters, only. Hence, it does not matter whether we read a before or after we write $b \neq a$. Note that we also require such a in the content x (preceded by forgettable letters) when applying the actions from the right-hand side of the equations (2) and (3). Finally, in equation (4) the leading write action a ensures that the partially lossy queue contains an a after application of $[\![a]\!]$. Then it does not matter whether we apply $[\![ta\overline{a}]\!]$ or $[\![t\overline{a}a]\!]$ to xa.

Now, consider the case $\mathcal{L} = (\varnothing, U)$ (i.e., we consider a reliable queue). Then the equations (1) and (2) are similar to the ones from Lemma 4.6.2 concerning the behavioral equivalence of a reliable queue. Additionally, (3) of Lemma 4.6.2 corresponds to the remaining two equations. Note that Huschenbett et al. only have proven the special case $t = \varepsilon$. However, this special case also follows from our more general case.

Finally, consider the case $\mathcal{L} = (F, \varnothing)$ (i.e., we consider a fully lossy queue). Since we do not have any unforgettable letter $b \in U$, we also have no equation of type (3). In this case, this lemma also generalizes [Köc16, Lemma 3.8].

Proof. Towards the proof of the first equation, let $a, b \in A$ be two distinct letters and $x \in A^* \cup \{\bot\}$ be a queue content. First, we consider $[\![\overline{a}b]\!](x) \neq \bot$. Then there are words $y \in (F \setminus \{a\})^*$ and $z \in A^*$ such that $x = yaz$. Therefore, we obtain

$$[\![\overline{a}b]\!](x) = [\![\overline{a}b]\!](yaz) = [\![b]\!](z) = zb = [\![\overline{a}]\!](yazb) = [\![\overline{a}]\!](xb) = [\![b\overline{a}]\!](x)\,.$$

Now, assume $[\![b\overline{a}]\!](x) \neq \bot$. Then there is a prefix ya of xb with $y \in (F \setminus \{a\})^*$. Since $a \neq b$ this prefix is proper. Hence, there is a $z \in A^*$ with $xb = yazb$. Then similarly we obtain $[\![b\overline{a}]\!](x) = [\![\overline{a}b]\!](x)$. This finally proves $[\![b\overline{a}]\!] = [\![\overline{a}b]\!]$ and, hence, $b\overline{a} \equiv \overline{a}b$.

Next we prove (2). Let $a, b \in A$ be two (not necessarily distinct) letters and $x \in A^*$. If $[\![\overline{a}a\overline{b}]\!](x) \neq \bot$, then there are words $y \in (F \setminus \{a\})^*$ and $z \in A^*$ with $x = yaz$. In this case we obtain

$$[\![\overline{a}a\overline{b}]\!](x) = [\![\overline{a}a\overline{b}]\!](yaz) = [\![\overline{b}]\!](za) = [\![\overline{a}\overline{b}]\!](yaza) = [\![\overline{a}\overline{b}]\!](xa) = [\![a\overline{a}\overline{b}]\!](x)\,.$$

Conversely, if $[\![a\overline{a}\overline{b}]\!](x) \neq \bot$ holds, there are words $y \in (F \setminus \{a\})^*$ and $z \in A^*$ with $xa = yaz$. From

$$\bot \neq [\![a\overline{a}\overline{b}]\!](x) = [\![\overline{a}\overline{b}]\!](xa) = [\![\overline{a}\overline{b}]\!](yaz) = [\![\overline{b}]\!](z)$$

we infer that $z \neq \varepsilon$, i.e., there is $z' \in A^*$ with $z = z'a$. Hence, we have $x = yaz'$. We finally obtain

$$[\![a\overline{a}\overline{b}]\!](x) = [\![\overline{b}]\!](z) = [\![\overline{b}]\!](z'a) = [\![a\overline{b}]\!](z') = [\![\overline{a}a\overline{b}]\!](yaz') = [\![\overline{a}a\overline{b}]\!](x)\,.$$

Finally, we show (3) and (4). So, let $a \in A$, $b \in U \cup \{a\}$, and $t, x \in A^*$. Again, we first assume $[\![bt\overline{a}a]\!](x) \neq \bot$. Then we also have $[\![\overline{a}a]\!](xbt) = [\![bt\overline{a}a]\!](x) \neq \bot$. Hence, there are $y \in (F \setminus \{a\})^*$ and $z \in A^*$ with $xbt = yaz$. Then we obtain

$$\begin{aligned}[\![bt\overline{a}a]\!](x) &= [\![\overline{a}a]\!](xbt) = [\![\overline{a}a]\!](yaz) = [\![a]\!](z) = za\\ &= [\![\overline{a}]\!](yaza) = [\![\overline{a}]\!](xbta) = [\![bta\overline{a}]\!](x)\,.\end{aligned}$$

Conversely, assume $[\![bta\overline{a}]\!](x) \neq \bot$. Then there is a prefix ya of $xbta$ with $y \in (F \setminus \{a\})^*$. By $b \in U \cup \{a\}$, the word ya is a prefix of xb. Then there is $z \in A^*$ with $xb = yaz$. We obtain

$$\begin{aligned}[\![bta\overline{a}]\!](x) &= [\![\overline{a}]\!](xbta) = [\![\overline{a}]\!](yazta) = zta\\ &= [\![a]\!](zt) = [\![\overline{a}a]\!](yazt) = [\![\overline{a}a]\!](xbt) = [\![bt\overline{a}a]\!](x)\,.\end{aligned}$$

This implies $[\![bta\overline{a}]\!] = [\![bt\overline{a}a]\!]$ and, therefore, our claims (3) and (4). ◂

Note that all of the equations from Lemma 4.7.2 preserve the relative order of the write actions and of the read actions. Our next aim is to prove that this also holds for any pair $s, t \in \Sigma^*$ with $s \equiv t$. To this end, we recall the projections wrt and rd to write resp. read actions, i.e.,

$$\mathrm{wrt}(t) = \pi_A(t) \qquad \text{and} \qquad \overline{\mathrm{rd}(t)} = \pi_{\overline{A}}(t)$$

for each $t \in \Sigma^*$. For example, we have $\mathrm{wrt}(a\overline{a}\overline{b}a) = aa$ and $\mathrm{rd}(a\overline{a}\overline{b}a) = ab$.

To prove that the behavioral equivalence preserves the values of these two projections, we need the following lemma, which allows to de-shuffle the projections of an action sequence $t \in \Sigma^*$ whenever $[\![t]\!](x)$ is defined.

Lemma 4.7.3. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $x \in A^*$, and $t \in \Sigma^*$ with $[\![t]\!](x) \neq \bot$. Then $[\![t]\!](x) = [\![\mathrm{wrt}(t)\overline{\mathrm{rd}(t)}]\!](x)$.*

Proof. Let $a \in A$ be a letter with $[\![\overline{a}a]\!](x) \neq \bot$. Then there are $y \in (F \setminus \{a\})^*$ and $z \in A^*$ with $x = yaz$. We have

$$[\![\overline{a}a]\!](x) = [\![\overline{a}a]\!](yaz) = [\![a]\!](z) = za = [\![\overline{a}]\!](yaza) = [\![\overline{a}]\!](xa) = [\![a\overline{a}]\!](x)\,,$$

i.e., we can commute $\overline{a}$ and a in this case. Similarly, we can commute $\overline{a}$ and b for two distinct letters $a, b \in A$ due to Lemma 4.7.2(1). Induction on the (minimal) number of transpositions needed to transform t into $\mathrm{wrt}(t)\overline{\mathrm{rd}(t)}$ finally yields the statement of our lemma. ◂

With this lemma we can prove the aforementioned preservation of the projections under equivalence:

Proposition 4.7.4. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F|+|U| \geq 2$ and $s, t \in \Sigma^*$ with $s \equiv t$. Then we have $\mathrm{wrt}(s) = \mathrm{wrt}(t)$ and $\mathrm{rd}(s) = \mathrm{rd}(t)$.*

Proof. We first prove $\mathrm{rd}(s) = \mathrm{rd}(t)$. By symmetry we can assume $|\mathrm{rd}(s)| \leq |\mathrm{rd}(t)|$. If $\mathrm{rd}(t) = \varepsilon$ then this implies immediately $\mathrm{rd}(s) = \mathrm{rd}(t)$. So, from now on we can assume that $\mathrm{rd}(t) \neq \varepsilon$ holds. Then there is $a \in A$ with $\mathrm{rd}(t) \in A^*a$. Since $|A| = |F| + |U| \geq 2$, there is another letter $b \in A \setminus \{a\}$. We have

$$\begin{aligned}
\bot \neq b^{|t|}\mathrm{wrt}(s) &= [\![s]\!](\mathrm{rd}(s)b^{|t|}) \\
&= [\![t]\!](\mathrm{rd}(s)b^{|t|}) && \text{(since } s \equiv t\text{)} \\
&= [\![\mathrm{wrt}(t)\overline{\mathrm{rd}(t)}]\!](\mathrm{rd}(s)b^{|t|}) && \text{(by Lemma 4.7.3)} \\
&= [\![\overline{\mathrm{rd}(t)}]\!](\mathrm{rd}(s)b^{|t|}\mathrm{wrt}(t)) =: x\,.
\end{aligned}$$

Since $|\mathrm{rd}(s)| \leq |\mathrm{rd}(t)|$, the word x is a suffix of $b^{|t|}\mathrm{wrt}(t)$. Suppose x is a proper suffix of $b^{|t|}\mathrm{wrt}(t)$. When reading $\mathrm{rd}(t)$ from the queue with content $\mathrm{rd}(s)b^{|t|}\mathrm{wrt}(t)$, the last letter to read is $a \neq b$. It follows that the result x is a proper suffix of $\mathrm{wrt}(t)$. But then we have

$$|b^{|t|}\mathrm{wrt}(s)| = |x| < |\mathrm{wrt}(t)| \leq |t| \leq |b^{|t|}\mathrm{wrt}(s)|\,,$$

which is a contradiction. Hence, x is not a proper suffix, i.e., $x = b^{|t|}\mathrm{wrt}(t)$. But then

$$[\![\overline{\mathrm{rd}(t)}]\!](\mathrm{rd}(s)b^{|t|}\mathrm{wrt}(t)) = x = b^{|t|}\mathrm{wrt}(t)$$

implies $[\![\overline{\mathrm{rd}(t)}]\!](\mathrm{rd}(s)) = \varepsilon \neq \bot$. Since $|\mathrm{rd}(s)| \leq |\mathrm{rd}(t)|$ holds, this is only possible if $\mathrm{rd}(s) = \mathrm{rd}(t)$.

Finally, the equation $\mathrm{wrt}(s) = \mathrm{wrt}(t)$ follows easily:

$$\begin{aligned}
\mathrm{wrt}(s) &= [\![s]\!](\mathrm{rd}(s)) \\
&= [\![t]\!](\mathrm{rd}(t)) && \text{(since } \mathrm{rd}(s) = \mathrm{rd}(t) \text{ and } s \equiv t\text{)} \\
&= \mathrm{wrt}(t)\,.
\end{aligned}$$

◂

Note that the converse implication of Proposition 4.7.4 does not hold in general. In Example 4.7.1 we have seen a counterexample. However, from the statements in the following subsection we will obtain a third property such that any pair of action sequences behave equivalently whenever they have the same projections and satisfy this third property.

4.7.2 Normal Forms

Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ (recall that plqs with underlying lossiness alphabet with $|F| + |U| = 1$ are essentially counters or stacks, respectively). In this subsection we prove that $\equiv$ is the least congruence on the free monoid satisfying the equations from Lemma 4.7.2. This is achieved by constructing a terminating and confluent semi-Thue system from those equations and by showing that every equivalence class of $\equiv$ contains a unique, irreducible word (we call this word the normal form). Later we will often use this normal form instead of the corresponding equivalence class wrt. $\equiv$.

Ordering the equations from Lemma 4.7.2 from left to right, we obtain an infinite semi-Thue system $\mathfrak{R}_{\mathcal{L}}$ consisting of the following rules for $a, b \in A$, and $t \in A^*$:

(1) $b\overline{a} \to \overline{a}b$ if $a \neq b$,

(2) $a\overline{a}\overline{b} \to \overline{a}a\overline{b}$,

(3) $bta\overline{a} \to bt\overline{a}a$ if $b \in U$, and

(4) $ata\overline{a} \to at\overline{a}a$.

Hence, the idea of this semi-Thue system is to pull read operations to the left as long as the equations from Lemma 4.7.2 permit.

First, we prove that $\mathfrak{R}_{\mathcal{L}}$ is terminating and confluent:

Lemma 4.7.5. *Let $\mathcal{L}$ be a lossiness alphabet. Then the semi-Thue system $\mathfrak{R}_{\mathcal{L}}$ is terminating and confluent.*

Proof. To prove termination we order the alphabet Σ such that $\overline{a} < b$ holds for each $a, b \in A$. Then we see that for any rule $\ell \to r$ from $\mathfrak{R}_{\mathcal{L}}$ the word r is length-lexicographically[viii] properly smaller than ℓ. Since this ordering is well-founded the semi-Thue system is terminating.

Due to termination of $\mathfrak{R}_{\mathcal{L}}$ it suffices to show that it is locally confluent. This is trivial if the left-hand sides of the applied rules do not overlap. Hence, we only have to consider the case of overlapping left-hand sides. To this end, let $a, b \in A$, $c, d \in U \cup \{a\}$, and $s, t \in A^*$. Then we see:

- $ct(\overline{a}a\overline{b}) \overset{(2)}{\Longleftarrow}_{\mathfrak{R}_{\mathcal{L}}} ct(a\overline{a}\overline{b}) = (cta\overline{a})\overline{b} \overset{(3)/(4)}{\Longrightarrow}_{\mathfrak{R}_{\mathcal{L}}} (ct\overline{a}a)\overline{b}$.
- $(csdt\overline{a}a) \overset{(3)/(4)}{\Longleftarrow}_{\mathfrak{R}_{\mathcal{L}}} (csdta\overline{a}) = cs(dta\overline{a}) \overset{(3)/(4)}{\Longrightarrow}_{\mathfrak{R}_{\mathcal{L}}} cs(dt\overline{a}a)$.

Therefore, $\mathfrak{R}_{\mathcal{L}}$ is confluent. ◂

[viii] A word $v \in \Sigma^*$ is *length-lexicographically* properly smaller than $w \in \Sigma^*$ if $|v| < |w|$ or $|v| = |w|$ and there are $\alpha, \beta \in \Sigma$ and a prefix $p \in \Sigma^*$ with $v \in p\alpha\Sigma^*$, $w \in p\beta\Sigma^*$, and $\alpha < \beta$.

Let $t \in \Sigma^*$ be an action sequence. Since the semi-Thue system is terminating and confluent, there is a unique irreducible word $\mathrm{nf}(t) \in \Sigma^*$ with $t \Rightarrow^*_{\mathfrak{R}_{\mathcal{L}}} \mathrm{nf}(t)$, the so-called *normal form* of t. Note that we have $t \equiv \mathrm{nf}(t)$ due to the equations from Lemma 4.7.2. According to the rules of $\mathfrak{R}_{\mathcal{L}}$ we obtain

$$\mathrm{NF}_{\mathcal{Q}_{\mathcal{L}}} := \overline{A}^* \cdot \Big(\bigcup_{a \in A} (F \setminus \{a\})^* a \overline{a} \Big)^* \cdot A^*$$

since these are precisely those words that do not contain any rule's left-hand side as a factor.

Because of the shape of the irreducible word $\mathrm{nf}(t)$ there are $k \in \mathbb{N}$, letters $a_i \in A$, words $x, z \in A^*$, and $y_i \in (F \setminus \{a_i\})^*$ (for $1 \leq i \leq k$) such that

$$\mathrm{nf}(t) = \overline{x} \cdot (y_1 a_1 \overline{a_1})(y_2 a_2 \overline{a_2}) \ldots (y_k a_k \overline{a_k}) \cdot z$$

holds. Note that k, a_i, x, y_i, and z are uniquely defined. We define

$$\mathrm{rd}_1(t) := x\,, \qquad \mathrm{rd}_2(t) := a_1 \ldots a_k\,, \qquad \mathrm{wrt}_1(t) := y_1 a_1 \ldots y_k a_k\,, \text{ and} \qquad \mathrm{wrt}_2(t) := z\,.$$

Then we see $\mathrm{rd}(t) = \mathrm{rd}(\mathrm{nf}(t)) = \mathrm{rd}_1(t)\mathrm{rd}_2(t)$ and $\mathrm{wrt}(t) = \mathrm{wrt}(\mathrm{nf}(t)) = \mathrm{wrt}_1(t)\mathrm{wrt}_2(t)$. Note that in contrast to the reliable case, in general we do not have $\mathrm{rd}_2(t) = \mathrm{wrt}_1(t)$. However, we can see that $\mathrm{rd}_2(t)$ is a special $\mathcal{L}$-subword of $\mathrm{wrt}_1(t)$. Concretely, we observe that $\mathrm{wrt}_1(t)$ is a reduced $\mathcal{L}$-superword of $\mathrm{rd}_2(t)$ (cf. Definition 4.4.6).

We can see that $\mathrm{rd}_1(t)$ can be obtained from $\mathrm{rd}(t)$ and $\mathrm{rd}_2(t)$ since $\mathrm{rd}_1(t)$ is the complementary prefix of $\mathrm{rd}(t)$ with respect to $\mathrm{rd}_2(t)$. Moreover, we obtain $\mathrm{wrt}_1(t)$ from $\mathrm{wrt}(t)$ and $\mathrm{rd}_2(t)$ since $\mathrm{wrt}_1(t)$ is the only prefix of $\mathrm{wrt}(t)$ which also is a reduced $\mathcal{L}$-superword of $\mathrm{rd}_2(t)$. Finally, we obtain $\mathrm{wrt}_2(t)$ from $\mathrm{wrt}(t)$ and $\mathrm{wrt}_1(t)$ since $\mathrm{wrt}_2(t)$ is the complementary suffix of $\mathrm{wrt}(t)$ with respect to $\mathrm{wrt}_1(t)$. So, the normal form of t is fully determined by the triple

$$\chi(t) := (\mathrm{wrt}(t), \mathrm{rd}(t), \mathrm{rd}_2(t))$$

which we call the *characteristic* of t.

Towards better readability we also write $\langle\!\langle y_1 a_1 \ldots y_k a_k, \overline{a_1 \ldots a_k} \rangle\!\rangle$ instead of the shuffled word $y_1 a_1 \overline{a_1} \ldots y_k a_k \overline{a_k}$. In this case the normal form of t can be written as follows:

$$\mathrm{nf}(t) = \overline{\mathrm{rd}_1(t)} \, \langle\!\langle \mathrm{wrt}_1(t), \overline{\mathrm{rd}_2(t)} \rangle\!\rangle \, \mathrm{wrt}_2(t)\,.$$

Note that for the words ab and $\overline{a}$ there is no such shuffled word $\langle\!\langle ab, \overline{a} \rangle\!\rangle$. Concretely, $\langle\!\langle w, \overline{r} \rangle\!\rangle$ is well-defined if, and only if, $w \in \mathrm{redsup}_{\mathcal{L}}(r)$ holds.

We can also see the shuffled word $\langle\!\langle \mathrm{wrt}_1(t), \overline{\mathrm{rd}_2(t)} \rangle\!\rangle$ as some kind of an overlap of the write and read actions in this normal form $\mathrm{nf}(t)$. Since this overlap is completely described by $\mathrm{wrt}(t)$ and $\mathrm{rd}_2(t)$, we also call $\mathrm{rd}_2(t)$ the *overlap* of t (cf. Figure 4.1).

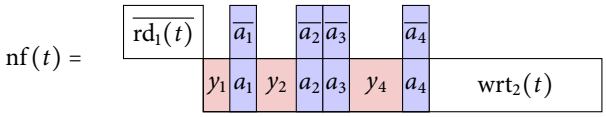

Figure 4.1. Visualization of the normal form of $t \in \Sigma^*$. The blue boxes represent the overlap $\mathrm{rd}_2(t)$ of t while the red and blue boxes represent $\mathrm{wrt}_1(t)$.

We can observe that for words $w, r \in A^*$ with $w \in \mathrm{redsup}_{\mathcal{L}}(r)$ the application of the transformation $[\![\langle\!\langle w, \overline{r} \rangle\!\rangle]\!]$ has no effect to the empty queue:

Observation 4.7.6. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $w, r \in A^*$ with $w \in \mathrm{redsup}_{\mathcal{L}}(r)$. Then we have $[\![\langle\!\langle w, \overline{r} \rangle\!\rangle]\!](\varepsilon) = \varepsilon$.*

Proof. By $w \in \mathrm{redsup}_{\mathcal{L}}(r)$ there are $a_1, \ldots, a_n \in A$ and $y_i \in (F \setminus \{a_i\})^*$ such that $r = a_1 \ldots a_n$ and $w = y_1 a_1 \ldots y_n a_n$. Then we have

$$\begin{aligned}
[\![\langle\!\langle w, \overline{r} \rangle\!\rangle]\!](\varepsilon) &= [\![(y_1 a_1 \overline{a_1})(y_2 a_2 \overline{a_2}) \ldots (y_n a_n \overline{a_n})]\!](\varepsilon) \\
&= [\![\overline{a_1}(y_2 a_2 \overline{a_2}) \ldots (y_n a_n \overline{a_n})]\!](y_1 a_1) \\
&= [\![(y_2 a_2 \overline{a_2}) \ldots (y_n a_n \overline{a_n})]\!](\varepsilon) \qquad \text{(since } y_1 \in (F \setminus \{a_1\})^*\text{)} \\
&\;\;\vdots \\
&= [\![\varepsilon]\!](\varepsilon) = \varepsilon\,.
\end{aligned}$$

◂

Remark 4.7.7. While $\mathrm{rd}_1(t)$ is defined using the semi-Thue system $\mathfrak{R}_{\mathcal{L}}$, it also has a natural meaning in terms of $[\![t]\!]$: from the shape of the normal form $\mathrm{nf}(t)$ we can infer that $\mathrm{rd}_1(t)$ is the shortest word $x \in A^*$ such that $[\![\mathrm{nf}(t)]\!](x)$ is defined. By Lemma 4.7.2 we have $t \equiv \mathrm{nf}(t)$. Hence, $\mathrm{rd}_1(t)$ is also the shortest word $x \in A^*$ such that $[\![t]\!](x)$ is defined. ⌟

Now, we show that the equations from Lemma 4.7.2 fully describe the equivalence classes of the behavioral equivalence $\equiv$:

Theorem 4.7.8. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ and $s, t \in \Sigma^*$. Then we have*

$$s \equiv t \iff \mathrm{nf}(s) = \mathrm{nf}(t)\,.$$

Proof. The implication "$\Leftarrow$" follows easily by two applications of Lemma 4.7.2:

$$s \equiv \mathrm{nf}(s) = \mathrm{nf}(t) \equiv t\,.$$

Now, we prove the converse implication "$\Rightarrow$". So let $s, t \in \Sigma^*$ with $s \equiv t$. By Proposition 4.7.4 we have $\mathrm{wrt}(s) = \mathrm{wrt}(t)$ and $\mathrm{rd}(s) = \mathrm{rd}(t)$. Since the normal forms of s and t are completely determined by their characteristics and since $\mathrm{rd}_2(s)$ is determined by $\mathrm{rd}(s)$ and $\mathrm{rd}_1(s)$, it suffices to prove $\mathrm{rd}_1(s) = \mathrm{rd}_1(t)$. For the following calculations let $r, w, z \in A^*$ such that $w \in \mathrm{redsup}_{\mathcal{L}}(r)$ and

$$\mathrm{nf}(s) = \overline{\mathrm{rd}_1(s)}\langle\!\langle w, \overline{r} \rangle\!\rangle z\,.$$

Then we get

$$\begin{aligned}
\bot \neq z &= [\![z]\!](\varepsilon) \\
&= [\![\langle\!\langle w, \overline{r}\rangle\!\rangle z]\!](\varepsilon) && \text{(by Observation 4.7.6)} \\
&= [\![\overline{\mathrm{rd}_1(s)}\langle\!\langle w, \overline{r}\rangle\!\rangle z]\!](\mathrm{rd}_1(s)) \\
&= [\![\mathrm{nf}(s)]\!](\mathrm{rd}_1(s)) \\
&= [\![s]\!](\mathrm{rd}_1(s)) && \text{(since } s \equiv \mathrm{nf}(s) \text{ by Lemma 4.7.2)} \\
&= [\![t]\!](\mathrm{rd}_1(s)) && \text{(since } s \equiv t\text{)} \\
&= [\![\mathrm{nf}(t)]\!](\mathrm{rd}_1(s))\,. && \text{(since } t \equiv \mathrm{nf}(t) \text{ by Lemma 4.7.2)}
\end{aligned}$$

Since $\overline{\mathrm{rd}_1(t)}$ is a prefix of $\mathrm{nf}(t)$ this implies $\bot \neq [\![\overline{\mathrm{rd}_1(t)}]\!](\mathrm{rd}_1(s))$. Since $\overline{\mathrm{rd}_1(t)}$ consists of read actions, only, this implies $|\mathrm{rd}_1(s)| \geq |\mathrm{rd}_1(t)|$. Now, by symmetry, we also obtain $|\mathrm{rd}_1(s)| \leq |\mathrm{rd}_1(t)|$ implying that these two words have the same length. But then $\bot \neq [\![\overline{\mathrm{rd}_1(t)}]\!](\mathrm{rd}_1(s))$ implies $\mathrm{rd}_1(s) = \mathrm{rd}_1(t)$ and therefore $\mathrm{nf}(s) = \mathrm{nf}(t)$. ◂

Remark 4.7.9. As a consequence, all words from the equivalence class of t share the same characteristic. Hence, this theorem allows us to speak of the characteristic of the equivalence class $[t]$ or transformation $[\![t]\!]$. With this characteristic in mind we can also apply wrt, rd, wrt_i, and rd_i (with $i = 1, 2$) to transformations from $\mathbb{T}(\mathcal{Q}_\mathcal{L})$ instead of words from Σ^*. ⌟

Now, consider an action sequence $t \in \Sigma^*$. Since $\mathfrak{R}_\mathcal{L}$ is terminating and confluent, we are able to compute the normal form of t. We can also see that each derivation in $\mathfrak{R}_\mathcal{L}$ starting with the word t has length at most $O(|t|^2)$ since we commute each pair of basic actions in t at most once. Hence, the computation of $\mathrm{nf}(t)$ is possible in polynomial time. From this fact, we can also infer the following statement:

Corollary 4.7.10. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$. Then the following* word problem *of $\mathbb{T}(\mathcal{Q}_\mathcal{L})$ is decidable in polynomial time:*

Input: *Two action sequences $s, t \in \Sigma^*$*
Question: *Does $s \equiv t$ hold?* ◂

In Chapter 5 we will consider some further questions in this direction, e.g., the so-called *rational membership problem* (does $[\![t]\!] \in L$ hold for a given action sequence $t \in \Sigma^*$ and a rational language $L \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$?), the *universality*, *inclusion*, and *emptiness of intersection* problem for rational languages in $\mathbb{T}(\mathcal{Q}_\mathcal{L})$.

4.7.3 Cancellation

From Theorem 4.7.8 we infer some statements about cancellation in the plq monoid. To this end, let $\mathcal{L} = (F, U)$ be a lossiness alphabet. From Lemma 4.7.2 we know that for $a, b \in A$ and $c \in U \cup \{a\}$ the equations

$$a\overline{ab} \equiv \overline{a}a\overline{b} \qquad \text{and} \qquad ca\overline{a} \equiv c\overline{a}a$$

hold. However, it is easy to see that $a\overline{a} \not\equiv \overline{a}a$ holds since

$$[\![a\overline{a}]\!](\varepsilon) = [\![\overline{a}]\!](a) = \varepsilon \neq \bot = [\![\overline{a}a]\!](\varepsilon)\,.$$

Hence, $\mathbb{T}(Q_{\mathcal{L}})$ is neither left- nor right-cancellative. However, $\mathbb{T}(Q_{\mathcal{L}})$ is cancellative with some restrictions according to the rules from $\mathfrak{R}_{\mathcal{L}}$:

Corollary 4.7.11. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$, $s, t \in \Sigma^*$, and $x, y \in A^*$. Then $\overline{x}sy \equiv \overline{x}ty$ implies $s \equiv t$.*

Proof. From the rules of the semi-Thue system $\mathfrak{R}_{\mathcal{L}}$ we infer $\mathrm{nf}(\overline{x}sy) = \overline{x}\,\mathrm{nf}(s)\,y$ and $\mathrm{nf}(\overline{x}ty) = \overline{x}\,\mathrm{nf}(t)\,y$. Hence from $\overline{x}sy \equiv \overline{x}ty$ we infer with Theorem 4.7.8:

$$\overline{x}\,\mathrm{nf}(s)\,y = \mathrm{nf}(\overline{x}sy) = \mathrm{nf}(\overline{x}ty) = \overline{x}\,\mathrm{nf}(t)\,y\,.$$

Since the free monoid Σ^* is cancellative, we obtain $\mathrm{nf}(s) = \mathrm{nf}(t)$. Again by Theorem 4.7.8 we infer $s \equiv t$. ◂

4.7.4 Green's Relations

We also learn from Theorem 4.7.8 that all of Green's relations are trivial in the transformation monoid $\mathbb{T}(Q_{\mathcal{L}})$. In other words, all of these relations coincide with the identity relation.

Corollary 4.7.12. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ and let $s, t \in \Sigma^*$ with $[\![s]\!]\,\mathrm{J}\,[\![t]\!]$. Then we have $s \equiv t$.*

Proof. Since $[\![t]\!] \in \mathbb{T}(Q_{\mathcal{L}}) \cdot [\![t]\!] \cdot \mathbb{T}(Q_{\mathcal{L}}) = \mathbb{T}(Q_{\mathcal{L}}) \cdot [\![s]\!] \cdot \mathbb{T}(Q_{\mathcal{L}})$ holds, there are action sequences $u, u' \in \Sigma^*$ such that $[\![u]\!]\,[\![s]\!]\,[\![u']\!] = [\![t]\!]$. Similarly, there are action sequences $v, v' \in \Sigma^*$ with $[\![v]\!]\,[\![t]\!]\,[\![v']\!] = [\![s]\!]$. Hence, we have

$$[\![s]\!] = [\![v]\!]\,[\![t]\!]\,[\![v']\!] = [\![vu]\!]\,[\![s]\!]\,[\![u'v']\!] = [\![vusu'v']\!]\,.$$

By Theorem 4.7.8 we have $\mathrm{nf}(s) = \mathrm{nf}(vusu'v')$ implying $|s| = |vusu'v'|$ and, hence, $vuu'v' = \varepsilon$. ◂

Hence, the relation J is the equality relation in $\mathbb{T}(Q_{\mathcal{L}})$. Since the equivalence relations L, R, H, and D are contained in the J-relation, all of these relations are trivial as well.

4.7.5 Composition

Next, we want to consider the normal form of the composition of two action sequences. With the help of this knowledge we gain some insight into the composition of two transformations from the plq monoid. Unfortunately, we still do not know a (simple) characterization of

$\mathrm{nf}(st)$ for arbitrary lossiness alphabets $\mathcal{L} = (F, U)$ and for arbitrary action sequences $s, t \in \Sigma^*$. Hence, we will only focus on the special case where $s \in A^*$ and $t \in \overline{A}^*$ holds.

First, let $\mathcal{L} = (\varnothing, U)$ be a lossiness alphabet without forgettable letters. In the previous section we have learned that the characteristic $\chi(w\overline{r})$ (where $r, w \in A^*$) consists of $\mathrm{wrt}(w\overline{r}) = w$, $\mathrm{rd}(w\overline{r}) = r$, and $\mathrm{rd}_2(w\overline{r})$ which is the longest suffix of r that also is a prefix of w (cf. Lemma 4.6.6). Recall that we have $\mathrm{rd}_2(w\overline{r}) = \mathrm{wrt}_1(w\overline{r})$ in this case.

Now, let $\mathcal{L} = (F, U)$ be an arbitrary lossiness alphabet. Then in the general case we do not have $\mathrm{rd}_2(w\overline{r}) = \mathrm{wrt}_1(w\overline{r})$. For example we have $\mathrm{wrt}_1(ab\overline{b}) = ab$ and $\mathrm{rd}_2(ab\overline{b}) = b$ if $a \in F \setminus \{b\}$ holds. However, we already know that $\mathrm{wrt}_1(w\overline{r}) \in \mathrm{redsup}_{\mathcal{L}}(\mathrm{rd}_2(w\overline{r}))$. We claim now that the characteristic $\chi(w\overline{r})$ consists of $\mathrm{wrt}(w\overline{r}) = w$, $\mathrm{rd}(w\overline{r}) = r$, and $\mathrm{rd}_2(w\overline{r})$ which is the longest suffix of r such that w starts with a reduced $\mathcal{L}$-superword of this suffix. But before we prove this claim, we want to give this special suffix of r a name:

Definition 4.7.13. Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $w, r \in A^*$. The *subword-suffix* $\mathrm{sws}(w, r)$ of w and r is the longest suffix r_2 of r such that there is a prefix w_1 of w with $w_1 \in \mathrm{redsup}_{\mathcal{L}}(r_2)$. ⌟

For example, let $a, b \in A$ be two distinct letters, $w = abba$, and $r = ba$. If $a, b \in F$ holds, then we have $\mathrm{sws}(w, r) = ba$ since $abba \in \mathrm{redsup}_{\mathcal{L}}(ba)$ holds in this case. Otherwise we have $\mathrm{sws}(w, r) = a$ since $w_1 \notin \mathrm{redsup}_{\mathcal{L}}(ba)$ for any prefix w_1 of w and $a \in \mathrm{redsup}_{\mathcal{L}}(a)$.

The following lemma proves that the word $\mathrm{rd}_2(w\overline{r})$ is exactly the subword-suffix of w and r.

Lemma 4.7.14. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ and $w, r \in A^*$. Then we have $\mathrm{rd}_2(w\overline{r}) = \mathrm{sws}(w, r)$.*

Proof. Since $\mathrm{rd}_1(w\overline{r})\mathrm{rd}_2(w\overline{r}) = \mathrm{rd}(w\overline{r}) = r$, the word $\mathrm{rd}_2(w\overline{r})$ is a suffix of r.

Let $w = w_1 w_2$ and $r = r_1 r_2$ be arbitrary factorizations with $w_1 \in \mathrm{redsup}_{\mathcal{L}}(r_2)$. Then there are letters $a_1, \ldots, a_k \in A$ and words $y_i \in (F \setminus \{a_i\})^*$ with $r_2 = a_1 a_2 \ldots a_k$ and $w_1 = y_1 a_1 y_2 a_2 \ldots y_k a_k$. We obtain then

$$\begin{aligned}
\bot \neq w_2 &= [\![\overline{a_1 a_2 \ldots a_k}]\!](y_1 a_1 y_2 a_2 \ldots y_k a_k\, w_2) && \text{(by Observation 4.7.6)}\\
&= [\![\overline{r_2}]\!](w_1 w_2) = [\![\overline{r_2}]\!](w)\\
&= [\![\overline{r}]\!](r_1 w)\\
&= [\![w\overline{r}]\!](r_1) = [\![\mathrm{nf}(w\overline{r})]\!](r_1)\,.
\end{aligned}$$

Since $\overline{\mathrm{rd}_1(w\overline{r})}$ is a prefix of $\mathrm{nf}(w\overline{r})$ we learn $[\![\overline{\mathrm{rd}_1(w\overline{r})}]\!](r_1) \neq \bot$ and therefore $|\mathrm{rd}_1(w\overline{r})| \leq |r_1|$. Then $\mathrm{rd}_1(w\overline{r})\mathrm{rd}_2(w\overline{r}) = r = r_1 r_2$ implies $|r_2| \leq |\mathrm{rd}_2(w\overline{r})|$. In particular, we have $|\mathrm{sws}(w, r)| \leq |\mathrm{rd}_2(w\overline{r})|$. Finally, the maximal length of $\mathrm{sws}(w, r)$ implies $\mathrm{rd}_2(w\overline{r}) = \mathrm{sws}(w, r)$. ◂

Let $\mathcal{L} = (\varnothing, U)$ be a lossiness alphabet without forgettable letters. In this case we already know $\mathrm{redsup}_{\mathcal{L}}(v) = \{v\}$ holds for all words $v \in \Sigma^*$. Hence, $\mathrm{sws}(w, r)$ is the longest suffix of r which is a prefix of w. So, in this special case Lemma 4.7.14 coincides with Lemma 4.6.6.

From Lemma 4.7.14 we obtain that for any action sequence $t \in \Sigma^*$ there is another action sequence $s \in \Sigma^*$ which behaves equivalently and is somewhat "simple":

Corollary 4.7.15. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ and $t \in \Sigma^*$. Then we have $t \equiv \overline{\mathrm{rd}_1(t)}\mathrm{wrt}(t)\overline{\mathrm{rd}_2(t)}$.*

Proof. From the definition of $\mathrm{nf}(t)$ we know that $\mathrm{wrt}_1(t) \in \mathrm{redsup}_{\mathcal{L}}(\mathrm{rd}_2(t))$ holds. Hence, we have

$$\mathrm{rd}_2(t) = \mathrm{sws}(\mathrm{wrt}_1(t), \mathrm{rd}_2(t)) = \mathrm{sws}(\mathrm{wrt}(t), \mathrm{rd}_2(t)) = \mathrm{rd}_2(\mathrm{wrt}(t)\overline{\mathrm{rd}_2(t)})$$

by Lemma 4.7.14. Therefore we have

$$\chi(\overline{\mathrm{rd}_1(t)}\mathrm{wrt}(t)\overline{\mathrm{rd}_2(t)}) = (\mathrm{wrt}(t), \mathrm{rd}_1(t)\mathrm{rd}_2(t), \mathrm{rd}_2(t)) = (\mathrm{wrt}(t), \mathrm{rd}(t), \mathrm{rd}_2(t)) = \chi(t)$$

which implies $t \equiv \overline{\mathrm{rd}_1(t)}\mathrm{wrt}(t)\overline{\mathrm{rd}_2(t)}$ by Theorem 4.7.8. ◂

We will use this result multiple times in the second part of this thesis when we consider the reachability problem in partially lossy queue automata.

Now, we want to consider the normal form of the composition of arbitrary action sequences $s, t \in \Sigma^*$. As we have mentioned before, we still do not know a good characterization for arbitrary lossiness alphabets. However, with Theorem 4.6.8 we have characterized $\mathrm{nf}(st)$ for lossiness alphabets $\mathcal{L} = (\varnothing, U)$ without forgettable letters. Additionally, the author presented a characterization of $\mathrm{nf}(st)$ for lossiness alphabets $\mathcal{L} = (F, \varnothing)$ without reliable letters in his Master's Thesis [Köc16]:

Theorem 4.7.16 ([Köc16]). *Let $\mathcal{L} = (F, \varnothing)$ be a lossiness alphabet with $|F| \geq 2$. Let $s, t \in \Sigma^*$ and $r = \mathrm{sws}(\mathrm{wrt}(s), \mathrm{rd}_2(s)\mathrm{rd}_1(t))$. Then we have*

$$\chi(st) = (\mathrm{wrt}(st), \mathrm{rd}(st), r\ \mathrm{rd}_2(t))\,.$$ ◂

4.7.6 Embeddings

Finally, we want to characterize which monoids embed into the plq monoid. Note that we omit the proofs of the results in this subsection. These can be found in [KKP18].

The following proposition characterizes the embeddings of arbitrary monoids into the plq monoid. This statement is also used in the proofs of all of the following theorems of this subsection.

Proposition 4.7.17 ([KKP18]). *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$, $\mathbb{M}$ be a monoid, $\phi\colon \mathbb{M} \hookrightarrow \mathbb{T}(Q_{\mathcal{L}})$ be an embedding, and $m, n \in \mathbb{M}$ such that $\mathrm{wrt}(\phi(m)) = \mathrm{wrt}(\phi(n))$ and $\mathrm{rd}(\phi(m)) = \mathrm{rd}(\phi(n))$ hold. Then there is $k \in \mathbb{M}$ with $kmk = knk$.* ◂

In other words, the images of two elements $m, n \in \mathbb{M}$ with the same projections to write and read actions can be equated with the help of another element $k \in \mathbb{M}$. From the proof in [KKP18] we obtain that this element k is either m^i or n^i for a (large) integer $i \in \mathbb{N}$.

From this proposition we infer (in a very complicated proof) a full characterization of embeddability of two partially lossy queue monoids:

Theorem 4.7.18 ([KKP18]). *Let $\mathcal{L}_1 = (F_1, U_1)$ and $\mathcal{L}_2 = (F_2, U_2)$ be two lossiness alphabets. Then $\mathbb{T}(Q_{\mathcal{L}_1})$ embeds into $\mathbb{T}(Q_{\mathcal{L}_2})$ if, and only if, all of the following properties hold:*

(a) $|F_1| \leq |F_2|$, *i.e., $\mathcal{L}_2$ has at least as many forgettable letters as $\mathcal{L}_1$.*

(b) If $|U_2| = 0$, *then also* $|U_1| = 0$, *i.e., if $\mathcal{L}_2$ consists of forgettable letters only, then so does $\mathcal{L}_1$.*

(c) If $|U_2| = 1$, *then* $|F_1| < |F_2|$ *or* $|U_1| \leq 1$, *i.e., if $\mathcal{L}_2$ has exactly one unforgettable letter and exactly as many forgettable letters as $\mathcal{L}_1$, then $\mathcal{L}_1$ contains at most one non-forgettable letter.* ◂

In Figure 4.2 we can find the complete picture of the embeddings stated in Theorem 4.7.18. In this picture the node (k, ℓ) represents the monoid $\mathbb{T}(Q_{(F_k, U_\ell)})$ where (F_k, U_ℓ) is a lossiness alphabet satisfying $|F_k| = k$ and $|U_\ell| = \ell$ (note that the plq monoids are isomorphic for two lossiness alphabets having the same numbers of forgettable resp. unforgettable letters). There is an edge from (k, ℓ) to (m, n) if, and only if, $\mathbb{T}(Q_{(F_k, U_\ell)})$ embeds into $\mathbb{T}(Q_{(F_m, U_n)})$. Note that we suppress those edges that follow from reflexivity and transitivity of the embedding relation.

We can also consider the trace monoids which embed into the plq monoid. Surprisingly, most of the partially lossy queue monoids embed the same set of trace monoids. These are the following ones:

Theorem 4.7.19 ([KKP18]). *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + 2|U| \geq 3$ and $\mathcal{A}$ be a distributed alphabet. Then the following statements are equivalent:*

(1) $\mathbb{M}(\mathcal{A})$ embeds into $\mathbb{T}(Q_{\mathcal{L}})$.

(2) $\mathbb{M}(\mathcal{A})$ embeds into $\{a, b\}^* \times \{c, d\}^*$.

(3) One of the following conditions holds:

(3.1) All nodes of $\mathcal{G}^c_{\mathcal{A}}$ have degree ≤ 1.

(3.2) The only non-trivial connected component $\mathcal{G}^c_{\mathcal{A}}$ is complete bipartite. ◂

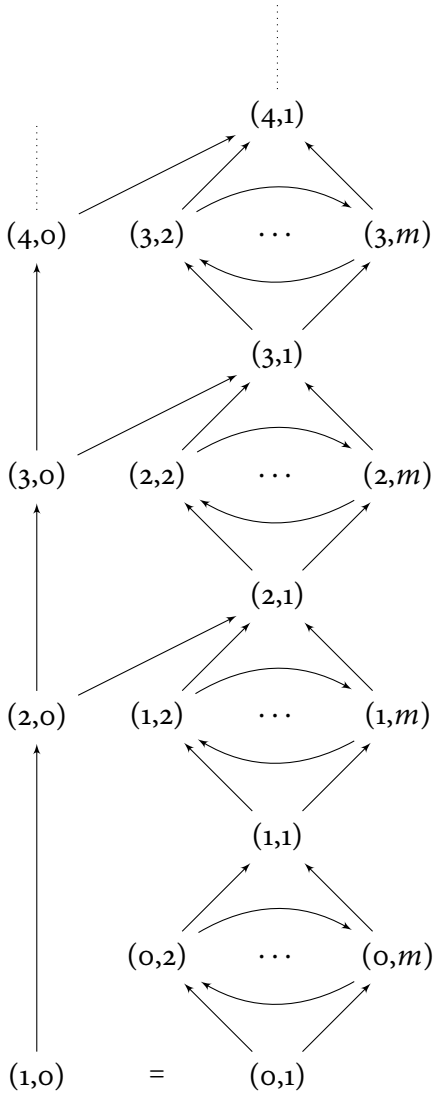

Figure 4.2. Visualization of Theorem 4.7.18.

Theorem 4.7.20 ([KKP18]). *Let $\mathcal{L} = (F, \varnothing)$ be a lossiness alphabet with $|F| = 2$ and $\mathcal{A}$ be a distributed alphabet. Then the following statements are equivalent:*

(1) $\mathbb{M}(\mathcal{A})$ embeds into $\mathbb{T}(\mathcal{Q}_{\mathcal{L}})$.

(2) One of the following conditions holds:

(2.1) All nodes of $\mathcal{G}^c_{\mathcal{A}}$ have degree ≤ 1.

(2.2) The only non-trivial connected component $\mathcal{G}^c_{\mathcal{A}}$ is a star graph. ◂

Note that in both theorems we have considered the complementary graph of the dependence graph of a distributed alphabet $\mathcal{A}$. We can see this graph as the independence graph of $\mathcal{A}$.

4.8 Distributed Queues

Finally, we consider the behavioral equivalence of distributed queues. We will characterize this relation with the help of a finite number of equations. Similar to the distributed stacks we will see that $\equiv_{\mathcal{Q}_{\mathcal{A}}}$ is closed under partial commutations. Hence, we can also understand action sequences as traces. From the equations we will derive a trace rewriting system. This trace rewriting system yields unique, irreducible traces - so-called normal forms. We also characterize the composition of two transformations.

Distributed queues consist of a finite number of queues with a specified synchronization mechanism and a finite number of counters without zero-tests (these correspond to the isolated nodes in the dependence graph). Hence, it is no surprise that the following results will be combinations of the statements from distributed stacks (cf. Section 4.5) and reliable queues (cf. Section 4.6).

A first (trivial) approach to analyze the behavioral equivalence of a distributed queue is to prove that $\equiv_{\mathcal{Q}_{\mathcal{A}}}$ is compatible with the projection lemma. In other words, two action sequences $s, t \in \Sigma^*$ behave equivalent (i.e., $s \equiv_{\mathcal{Q}_{\mathcal{A}}} t$) if, and only if, their projections to each process behave equivalent (i.e., $\pi_i(s) \equiv_{\mathcal{Q}_{A_i}} \pi_i(t)$ where π_i projects to the actions applicable to process i). Unfortunately, this equivalence does not hold as we will see later in Lemma 4.8.4. So, the analysis of $\equiv_{\mathcal{Q}_{\mathcal{A}}}$ is a bit more complicated as expected.

4.8.1 Basic Properties

First, we prove that the behavioral equivalence is compatible with partial commutations described by $\mathcal{A}$:

Lemma 4.8.1. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $a, b \in A$ with $a \parallel b$ be two independent letters, $\alpha \in \{a, \overline{a}\}$, and $\beta \in \{b, \overline{b}\}$. Then we have $\alpha\beta \equiv \beta\alpha$.*

Proof. The proof of this lemma is similar to the proof of Lemma 4.5.1(2). ◂

So, we can extend our underlying distributed alphabet $\mathcal{A} = (A, P, \mathrm{M})$ to another distributed alphabet $\mathcal{E} := (\Sigma, P, \mathrm{M}')$ on the set of all basic actions where $\mathrm{M}' := \mathrm{M} \cup \{(\overline{a}, i) \mid (a, i) \in \mathrm{M}\}$. For a trace $\tau \in \mathbb{M}(\mathcal{A})$ we denote $\overline{\tau} := [\overline{t}]$ for a word $t \in \tau$. For a trace language $L \subseteq \mathbb{M}(\mathcal{A})$ we also write $\overline{L} := \{\overline{\tau} \mid \tau \in L\}$.

We can observe that $\equiv$ is a quotient of the partial commutations $\approx_{\mathcal{E}}$. So, we are able to define the distributed queue's semantics $[\![.]\!]$ for traces. We define $[\![\tau]\!] := [\![t]\!]$ for a word $t \in \tau$ and any action trace $\tau \in \mathbb{M}(\mathcal{E})$. Note that this definition is well-defined according to Lemma 4.8.1. Moreover we write $\sigma \equiv_{Q_{\mathcal{A}}} \tau$ (or $\sigma \equiv \tau$ if the situation is clear) if, and only if, $[\![\sigma]\!] = [\![\tau]\!]$ holds for each pair of action traces $\sigma, \tau \in \mathbb{M}(\mathcal{E})$. Note that this is the case whenever we have $s \equiv_{Q_{\mathcal{A}}} t$ for action sequences $s \in \sigma$ and $t \in \tau$.

For action traces $\tau \in \mathbb{M}(\mathcal{E})$ we also introduce the projection π_i to the actions applicable to a process $i \in P$. Formally, in our situation this projection is the homomorphism $\pi_i \colon \mathbb{M}(\mathcal{E}) \to (A_i \cup \overline{A_i})^*$ satisfying $[\pi_i(\tau)] = \pi_{A_i \cup \overline{A_i}}(\tau)$. Note that this homomorphism is well-defined since $\mathbb{M}(\mathcal{E}{\restriction}_{A_i \cup \overline{A_i}})$ is isomorphic to $(A_i \cup \overline{A_i})^*$.

Now, recall the duality map $\mathrm{d} \colon \Sigma^* \to \Sigma^*$ which satisfies

$$\mathrm{d}(\varepsilon) = \varepsilon\,, \qquad \mathrm{d}(at) = \mathrm{d}(t)\overline{a}\,, \text{ and } \qquad \mathrm{d}(\overline{a}t) = \mathrm{d}(t)a$$

for each $t \in \Sigma^*$ and $a \in A$. We want to lift this function to traces. To this end, let $\alpha, \beta \in \Sigma$ be two basic actions with $\alpha \parallel \beta$. Then we have $\mathrm{d}(\alpha) \parallel \mathrm{d}(\beta)$ implying

$$\mathrm{d}(\alpha\beta) = \mathrm{d}(\beta)\,\mathrm{d}(\alpha) \approx_{\mathcal{E}} \mathrm{d}(\alpha)\,\mathrm{d}(\beta) = \mathrm{d}(\beta\alpha)$$

(recall that d is an antimorphism). Then the map $d \colon \mathbb{M}(\mathcal{E}) \to \mathbb{M}(\mathcal{E})$ with $\mathrm{d}([t]) = [\mathrm{d}(t)]$ for each $t \in \Sigma^*$ is well-defined. Additionally, this function is still a bijective antimorphism and an involution.

The following lemma proves that, if κ can be reached from λ via an action trace τ, then we can reach λ from κ (both in reversed order) via $\mathrm{d}(\tau)$.

Lemma 4.8.2. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\lambda, \kappa \in \mathbb{M}(\mathcal{A})$, and $\tau \in \mathbb{M}(\mathcal{E})$ with $[\![\tau]\!](\lambda) = \kappa$. Then we have $[\![\mathrm{d}(\tau)]\!](\kappa^{\mathrm{R}}) = \lambda^{\mathrm{R}}$.*

Proof. We prove this lemma by induction on the length of τ. First, let $\tau = \varepsilon$. Then we have $\lambda = \kappa$ and $[\![\mathrm{d}(\tau)]\!](\kappa^{\mathrm{R}}) = \kappa^{\mathrm{R}} = \lambda^{\mathrm{R}}$.

Now, let $\tau = \alpha\sigma$ for $\alpha \in \Sigma$ and $\sigma \in \mathbb{M}(\mathcal{E})$. We distinguish the following two cases:

(1) $\alpha = a \in A$. Then we have $\kappa = [\![a\sigma]\!](\lambda) = [\![\sigma]\!](\lambda a)$. By the induction hypothesis we infer $a\lambda^{\mathrm{R}} = [\![\mathrm{d}(\sigma)]\!](\kappa^{\mathrm{R}})$ implying

$$\lambda^{\mathrm{R}} = [\![\overline{a}]\!](a\lambda^{\mathrm{R}}) = [\![\overline{a}]\!]([\![\mathrm{d}(\sigma)]\!](\kappa^{\mathrm{R}})) = [\![\mathrm{d}(\sigma)\overline{a}]\!](\kappa^{\mathrm{R}}) = [\![\mathrm{d}(\tau)]\!](\kappa^{\mathrm{R}})\,.$$

(2) $\alpha = \overline{a} \in \overline{A}$. Since $\bot \neq \kappa = [\![\overline{a}\sigma]\!](\lambda) = [\![\sigma]\!]([\![\overline{a}]\!](\lambda))$ there is $\mu \in \mathbb{M}(\mathcal{A})$ with $\lambda = a\mu$. Hence, we have $\kappa = [\![\overline{a}\sigma]\!](\lambda) = [\![\sigma]\!](\mu)$. By the induction hypothesis we have $\mu^{\mathrm{R}} = [\![\mathrm{d}(\sigma)]\!](\kappa^{\mathrm{R}})$ which implies

$$\lambda^{\mathrm{R}} = \mu^{\mathrm{R}}a = [\![a]\!](\mu^{\mathrm{R}}) = [\![a]\!]([\![\mathrm{d}(\sigma)]\!](\kappa^{\mathrm{R}})) = [\![\mathrm{d}(\sigma)a]\!](\kappa^{\mathrm{R}}) = [\![\mathrm{d}(\tau)]\!](\kappa^{\mathrm{R}})\,.$$ ◂

From this lemma we obtain that d is compatible with the behavioral equivalence.

Corollary 4.8.3. *Let $\mathcal{A} = (A, P, M)$ be a distributed alphabet and $\sigma, \tau \in \mathbb{M}(\mathcal{E})$. Then we have $\sigma \equiv \tau$ if, and only if, $\mathrm{d}(\sigma) \equiv \mathrm{d}(\tau)$.*

Proof. First suppose $\sigma \equiv \tau$. Let $\kappa \in \mathbb{M}(\mathcal{A})$. Assume $[\![\mathrm{d}(\sigma)]\!](\kappa) \neq \bot$. Then there is $\lambda \in \mathbb{M}(\mathcal{A})$ with $\lambda = [\![\mathrm{d}(\sigma)]\!](\kappa)$. By $\mathrm{d}(\mathrm{d}(\sigma)) = \sigma$ and Lemma 4.8.2 we infer $[\![\sigma]\!](\lambda^{\mathrm{R}}) = [\![\mathrm{d}(\mathrm{d}(\sigma))]\!](\lambda^{\mathrm{R}}) = \kappa^{\mathrm{R}}$. From $\sigma \equiv \tau$ we know $[\![\tau]\!](\lambda^{\mathrm{R}}) = \kappa^{\mathrm{R}} \neq \bot$. By Lemma 4.8.2 we obtain $[\![\mathrm{d}(\tau)]\!](\kappa) = \lambda = [\![\mathrm{d}(\sigma)]\!](\kappa)$.

Now suppose $[\![\mathrm{d}(\tau)]\!](\kappa) \neq \bot$. Similarly, we can prove $[\![\mathrm{d}(\sigma)]\!](\kappa) = [\![\mathrm{d}(\tau)]\!](\kappa)$. This finally implies $[\![\mathrm{d}(\sigma)]\!] = [\![\mathrm{d}(\tau)]\!]$ and, hence, $\mathrm{d}(\sigma) \equiv \mathrm{d}(\tau)$.

Since d is an involution, this result also implies the converse implication. ◂

4.8.2 Normal Forms

Now, we are able to list a finite number of (context-sensitive) commutations that hold in the behavioral equivalence. We will see later that these equations fully describe this relation.

Lemma 4.8.4. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet. Then the following statements hold for $a, b \in A$:*

(1) $a\overline{b} \equiv \overline{b}a$ if $a \neq b$,

(2) $a\overline{a}\overline{b} \equiv \overline{a}a\overline{b}$ and $ba\overline{a} \equiv b\overline{a}a$ if $a \nparallel b$, and

(3) $a\overline{a} \equiv \varepsilon$ if $a \in \mathrm{Isolated}(\mathcal{A})$.

Note that the equations (1) and (2) are similar to the ones known from reliable queues (cf. Lemma 4.6.2 resp. [HKZ17]) and (3) is similar to the equation known from counters (cf. equation (1) from Lemma 4.4.1).

Proof. Towards the first equation, we assume $a \neq b$. Let $\kappa, \lambda \in \mathbb{M}(\mathcal{A})$ with $[\![a\overline{b}]\!](\lambda) = \kappa \neq \bot$. Then from the semantics of distributed queues we obtain $\lambda a = b\kappa$. Since $a \neq b$ holds, we obtain from Levi's lemma for traces (cf. Theorem 3.4.6) another trace $\mu \in \mathbb{M}(\mathcal{A})$ with $\lambda a = b\kappa = b\mu a$. Utilizing the cancellation property we get $\mu a = \kappa$ and $b\mu = \lambda$. Then we have

$$[\![a\overline{b}]\!](\lambda) = [\![a\overline{b}]\!](b\mu) = [\![\overline{b}]\!](b\mu a) = \mu a = [\![a]\!](\mu) = [\![\overline{b}a]\!](b\mu) = [\![\overline{b}a]\!](\lambda)\,.$$

Conversely, let $\kappa, \lambda \in \mathbb{M}(\mathcal{A})$ with $[\![\overline{b}a]\!](\lambda) = \kappa$. By definition there is $\mu \in \mathbb{M}(\mathcal{A})$ with $\lambda = b\mu$ (and $\kappa = \mu a$). This implies

$$[\![\overline{b}a]\!](\lambda) = [\![\overline{b}a]\!](b\mu) = [\![a]\!](\mu) = \mu a = [\![\overline{b}]\!](b\mu a) = [\![a\overline{b}]\!](b\mu) = [\![a\overline{b}]\!](\lambda)\,.$$

Hence, we infer $a\overline{b} \equiv \overline{b}a$.

Next, we prove $ba\overline{a} \equiv b\overline{a}a$ for $a \nparallel b$. First, let $\lambda \in \mathbb{M}(\mathcal{A})$ with $[\![ba\overline{a}]\!](\lambda) \neq \bot$. Then there is $\kappa \in \mathbb{M}(\mathcal{A})$ with $\lambda ba = a\kappa$. By the projection lemma (Theorem 3.4.4) we have $\pi_i(\lambda ba) = \pi_i(a\kappa)$ for each $i \in P$. Since $a \nparallel b$ there is $i \in a\,\mathrm{M} \cap b\,\mathrm{M} \subseteq P$. Then we have

$$\pi_i(\lambda)ba = \pi_i(\lambda ba) = \pi_i(a\kappa) = a\pi_i(\kappa)$$

implying the existence of a word $w \in A_i^*$ with $\pi_i(\lambda)ba = a\pi_i(\kappa) = awa$. Hence, there also is a trace $\mu \in \mathbb{M}(\mathcal{A})$ with $\lambda ba = a\kappa = a\mu a$. By the cancellation property we also learn $\lambda b = a\mu$. Then we see

$$[\![ba\overline{a}]\!](\lambda) = [\![\overline{a}]\!](\lambda ba) = [\![\overline{a}]\!](a\mu a) = \mu a = [\![a]\!](\mu) = [\![\overline{a}a]\!](a\mu) = [\![\overline{a}a]\!](\lambda b) = [\![b\overline{a}a]\!](\lambda)\,.$$

Conversely, let $\lambda \in \mathbb{M}(\mathcal{A})$ with $[\![b\overline{a}a]\!](\lambda) \neq \bot$. Then there is $\kappa \in \mathbb{M}(\mathcal{A})$ with $\lambda b = a\kappa$. We see

$$[\![b\overline{a}a]\!](\lambda) = [\![\overline{a}a]\!](\lambda b) = [\![\overline{a}a]\!](a\kappa) = [\![a]\!](\kappa) = \kappa a = [\![\overline{a}]\!](a\kappa a) = [\![\overline{a}]\!](\lambda ba) = [\![ba\overline{a}]\!](\lambda)\,.$$

Hence, we obtain $[\![ba\overline{a}]\!] = [\![b\overline{a}a]\!]$ implying $ba\overline{a} \equiv b\overline{a}a$. The equation $a\overline{a}\overline{b} \equiv \overline{a}a\overline{b}$ is dual to $ba\overline{a} \equiv b\overline{a}a$ and holds due to Corollary 4.8.3.

Finally, we assume $a \in \mathrm{Isolated}(\mathcal{A})$. Let $\lambda \in \mathbb{M}(\mathcal{A})$. Since a is an isolated vertex in the dependence graph we have $\lambda a = a\lambda$. This is due to $\pi_i(\lambda a) = \pi_i(a\lambda)$ for each $i \in P$ (this is the projection-lemma Theorem 3.4.4). Hence, we infer

$$[\![a\overline{a}]\!](\lambda) = [\![\overline{a}]\!](\lambda a) = [\![\overline{a}]\!](a\lambda) = [\![\varepsilon]\!](\lambda)\,. \qquad \blacktriangleleft$$

Our next aim is to prove that two equivalently behaving sequences σ and τ write and read the same subsequences of letters into each queue, except for the queues with a singleton alphabet (which are essentially counters). Before we do this, we first show that the application of an action trace $\tau \in \mathbb{M}(\mathcal{E})$ to a content $\lambda \in \mathbb{M}(\mathcal{A})$ does not end up in the error state $\bot$ if, and only if, this does not happen in any queue of our distributed data type $\mathcal{Q}_{\mathcal{A}}$:

Lemma 4.8.5. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\lambda \in \mathbb{M}(\mathcal{A})$, and $\tau \in \mathbb{M}(\mathcal{E})$. Then we have $[\![\tau]\!]_{\mathcal{Q}_{\mathcal{A}}}(\lambda) \neq \bot$ if, and only if, for each $i \in P$ we have $[\![\pi_i(\tau)]\!]_{\mathcal{Q}_{A_i}}(\pi_i(\lambda)) \neq \bot$.*

Proof. We prove this statement by induction on the length $|\tau|$ of τ. The case $\tau = \varepsilon$ is obvious since we have $[\![\varepsilon]\!]_{\mathcal{Q}_{\mathcal{A}}}(\lambda) = \lambda \neq \bot$ and $[\![\varepsilon]\!]_{\mathcal{Q}_{A_i}}(\pi_i(\lambda)) = \pi_i(\lambda) \neq \bot$.

Now let $\tau = \alpha\sigma$ where $\alpha \in \Sigma$ and $\sigma \in \mathbb{M}(\mathcal{E})$. We have to distinguish two cases:

(1) $\alpha = a \in A$. Then we have $[\![a\sigma]\!]_{\mathcal{Q}_{\mathcal{A}}}(\lambda) = [\![\sigma]\!]_{\mathcal{Q}_{\mathcal{A}}}(\lambda a)$ and

$$[\![\pi_i(a\sigma)]\!]_{\mathcal{Q}_{A_i}}(\pi_i(\lambda)) = [\![\pi_i(\sigma)]\!]_{\mathcal{Q}_{A_i}}(\pi_i(\lambda a))\,.$$

By induction hypothesis we have $[\![\sigma]\!]_{\mathcal{Q}_{\mathcal{A}}}(\lambda a) \neq \bot$ if, and only if, $[\![\pi_i(\sigma)]\!]_{\mathcal{Q}_{A_i}}(\pi_i(\lambda a)) \neq \bot$ and we are done.

(2) $\alpha = \overline{a} \in \overline{A}$. First, we assume $[\![\overline{a}\sigma]\!]_{\mathcal{Q}_{\mathcal{A}}}(\lambda) \neq \bot$. Then there is $\kappa \in \mathbb{M}(\mathcal{A})$ with $\lambda = a\kappa$ and

$$\bot \neq [\![\overline{a}\sigma]\!]_{\mathcal{Q}_{\mathcal{A}}}(\lambda) = [\![\sigma]\!]_{\mathcal{Q}_{\mathcal{A}}}(\kappa)\,.$$

Now, let $i \in P$. Then, by the induction hypothesis, we have $[\![\pi_i(\sigma)]\!]_{\mathcal{Q}_{A_i}}(\pi_i(\kappa)) \neq \bot$. We also know $\pi_i(\lambda) = \pi_i(a)\pi_i(\kappa)$ (since π_i is a homomorphism). This implies

$$[\![\pi_i(\tau)]\!]_{\mathcal{Q}_{A_i}}(\pi_i(\lambda)) = [\![\pi_i(\overline{a}\sigma)]\!]_{\mathcal{Q}_{A_i}}(\pi_i(a\kappa)) = [\![\pi_i(\sigma)]\!]_{\mathcal{Q}_{A_i}}(\pi_i(\kappa)) \neq \bot .$$

Conversely, let $[\![\pi_i(\overline{a}\sigma)]\!]_{\mathcal{Q}_{A_i}}(\pi_i(\lambda)) \neq \bot$ for each $i \in P$. Then for each $i \in P$ there is $w_i \in A_i^*$ with $\pi_i(\lambda) = \pi_i(aw_i)$. Using the projection-lemma (Theorem 3.4.4) and by the cancellation property of $\mathbb{M}(\mathcal{A})$ we obtain a trace $\kappa \in \mathbb{M}(\mathcal{A})$ with $\lambda = a\kappa$ and $\pi_i(\kappa) = w_i$. By definition of the semantics of $\overline{a}$ we learn $\bot \neq [\![\pi_i(\overline{a}\sigma)]\!]_{\mathcal{Q}_{A_i}}(\pi_i(\lambda)) = [\![\pi_i(\sigma)]\!]_{\mathcal{Q}_{A_i}}(\pi_i(\kappa))$ for each $i \in P$. Then from the induction hypothesis we infer

$$\bot \neq [\![\sigma]\!]_{\mathcal{Q}_{\mathcal{A}}}(\kappa) = [\![\overline{a}\sigma]\!]_{\mathcal{Q}_{\mathcal{A}}}(\lambda) . \qquad \blacktriangleleft$$

Now, we shall generalize the projections to write and read actions: $\mathrm{wrt}, \mathrm{rd}\colon \mathbb{M}(\mathcal{E}) \to \mathbb{M}(\mathcal{A})$ with $\mathrm{wrt}(\tau) = \pi_A(\tau)$ and $\overline{\mathrm{rd}(\tau)} = \pi_{\overline{A}}(\tau)$. In other words, both projections are natural extensions of the projections from Section 4.6 to traces of actions.

Before we show that two equivalently behaving action traces agree in their projections, we first have to prove that any non-failing computation leads to the same result as a de-shuffled one.

Lemma 4.8.6. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\lambda \in \mathbb{M}(\mathcal{A})$, and $\tau \in \mathbb{M}(\mathcal{E})$ with $[\![\tau]\!](\lambda) \neq \bot$. Then we have $[\![\tau]\!](\lambda) = [\![\mathrm{wrt}(\tau)\overline{\mathrm{rd}(\tau)}]\!](\lambda)$.*

Proof. First, let $a, b \in A$ with $[\![\overline{a}b]\!](\lambda) \neq \bot$. If $a \neq b$ we have $\overline{a}b \equiv b\overline{a}$ by Lemma 4.8.4(1) implying $[\![\overline{a}b]\!](\lambda) = [\![b\overline{a}]\!](\lambda)$. So, assume $a = b$ from now on. By $[\![\overline{a}a]\!](\lambda) \neq \bot$ we know that there exists a $\kappa \in \mathbb{M}(\mathcal{A})$ with $\lambda = a\kappa$. Then we obtain

$$[\![\overline{a}a]\!](\lambda) = [\![\overline{a}a]\!](a\kappa) = [\![a]\!](\kappa) = \kappa a = [\![\overline{a}]\!](a\kappa a) = [\![a\overline{a}]\!](a\kappa) = [\![a\overline{a}]\!](\lambda) .$$

By induction on the (minimal) number of transpositions needed to transform the action trace τ into $\mathrm{wrt}(\tau)\overline{\mathrm{rd}(\tau)}$ we obtain our claim. $\blacktriangleleft$

With the help of the prior lemmata we will prove the aforementioned statement that the behavioral equivalence preserves the relative order of read and write actions - at least for the non-trivial processes. To simplify notations we introduce the projections to write and read actions of a given process $i \in P$: $\mathrm{wrt}_i := \mathrm{wrt} \circ \pi_i = \pi_{A_i}\colon \mathbb{M}(\mathcal{E}) \to A_i^*$ and $\mathrm{rd}_i := \mathrm{rd} \circ \pi_i\colon \mathbb{M}(\mathcal{E}) \to A_i^*$ (recall that $\mathbb{M}(\mathcal{E}\restriction_{A_i})$ is isomorphic to A_i^* and $\mathbb{M}(\mathcal{E}\restriction_{\overline{A_i}})$ is isomorphic to $\overline{A_i}^*$).

Proposition 4.8.7. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\sigma, \tau \in \mathbb{M}(\mathcal{E})$ with $\sigma \equiv \tau$, and $i \in P$ such that $A_i \notin \mathrm{Isolated}(\mathcal{A})$. Then we have $\mathrm{wrt}_i(\sigma) = \mathrm{wrt}_i(\tau)$ and $\mathrm{rd}_i(\sigma) = \mathrm{rd}_i(\tau)$.*

Note that $\sigma \equiv_{\mathcal{Q}_\mathcal{A}} \tau$ does not imply $\pi_i(\sigma) \equiv_{\mathcal{Q}_{A_i}} \pi_i(\tau)$ for each $i \in P$: for example, let $\mathcal{A} = (A, P, \mathrm{M})$ with $A = \{a, b\}$, $P = \{1, 2\}$, and $\mathrm{M} = \{(a,1), (a,2), (b,2)\}$. Then from Lemma 4.8.4 we learn $a\overline{a}\overline{b} \equiv_{\mathcal{Q}_\mathcal{A}} \overline{a}a\overline{b}$. However, from Theorem 4.6.3 we know $a\overline{a} \not\equiv_{\mathcal{Q}_{A_1}} \overline{a}a$. Hence, this proposition is no corollary of Theorem 4.6.3.

Proof. First, we prove that $\mathrm{rd}_i(\sigma)$ is a prefix of $\mathrm{rd}_i(\tau)$ if $|A_i| \geq 2$. Towards a contradiction, we suppose that $\mathrm{rd}_i(\sigma)$ is not a prefix of $\mathrm{rd}_i(\tau)$. Since $|A_i| \geq 2$ holds, there is $b \in A_i$ such that $\mathrm{rd}_i(\sigma)$ is no prefix of $\mathrm{rd}_i(\tau)b^n$ where $n := |\mathrm{rd}_i(\sigma)|$. Then we have:

$$\begin{aligned} \bot \neq b^n \mathrm{wrt}(\tau) &= [\![\tau]\!]_{\mathcal{Q}_\mathcal{A}}(\mathrm{rd}(\tau)b^n) \\ &= [\![\sigma]\!]_{\mathcal{Q}_\mathcal{A}}(\mathrm{rd}(\tau)b^n) && \text{(since } \sigma \equiv \tau\text{)} \\ &= [\![\mathrm{wrt}(\sigma)\overline{\mathrm{rd}(\sigma)}]\!]_{\mathcal{Q}_\mathcal{A}}(\mathrm{rd}(\tau)b^n)\,. && \text{(by Lemma 4.8.6)} \end{aligned}$$

Then due to Lemma 4.8.5 we also have $[\![\mathrm{wrt}_i(\sigma)\overline{\mathrm{rd}_i(\sigma)}]\!]_{\mathcal{Q}_{A_i}}(\mathrm{rd}_i(\tau)b^n) \neq \bot$. This implies

$$\bot \neq [\![\mathrm{wrt}_i(\sigma)\overline{\mathrm{rd}_i(\sigma)}]\!]_{\mathcal{Q}_{A_i}}(\mathrm{rd}_i(\tau)b^n) = [\![\overline{\mathrm{rd}_i(\sigma)}]\!]_{\mathcal{Q}_{A_i}}(\mathrm{rd}_i(\tau)b^n\,\mathrm{wrt}_i(\sigma))\,.$$

Hence, there is $w \in A_i^*$ with $\mathrm{rd}_i(\sigma)w = \mathrm{rd}_i(\tau)b^n\,\mathrm{wrt}_i(\sigma)$. Since $n = |\mathrm{rd}_i(\sigma)| \leq |\mathrm{rd}_i(\tau)| + n$ we infer that $\mathrm{rd}_i(\sigma)$ is a prefix of $\mathrm{rd}_i(\tau)b^n$. This is a contradiction to the choice of $b \in A_i$. Hence, $\mathrm{rd}_i(\sigma)$ is a prefix of $\mathrm{rd}_i(\tau)$. By symmetry, $\mathrm{rd}_i(\tau)$ also is a prefix of $\mathrm{rd}_i(\sigma)$ implying $\mathrm{rd}_i(\sigma) = \mathrm{rd}_i(\tau)$.

Now, let $A_i = \{a\}$ with $a \notin \mathrm{Isolated}(\mathcal{A})$. Then there is $j \in P$ with $A_i \subsetneq A_j$. We already know $\mathrm{rd}_j(\sigma) = \mathrm{rd}_j(\tau)$. Then we infer the following:

$$\mathrm{rd}_i(\sigma) = \mathrm{rd}_i(\mathrm{rd}_j(\sigma)) = \mathrm{rd}_i(\mathrm{rd}_j(\tau)) = \mathrm{rd}_i(\tau)\,.$$

Finally, we have to prove $\mathrm{wrt}_i(\sigma) = \mathrm{wrt}_i(\tau)$. By Corollary 4.8.3 we know $\mathrm{d}(\sigma) \equiv \mathrm{d}(\tau)$. Then using the result from above yields

$$\mathrm{wrt}_i(\sigma) = \mathrm{d}\big(\overline{\mathrm{rd}_i(\mathrm{d}(\sigma))}\big) = \mathrm{d}\big(\overline{\mathrm{rd}_i(\mathrm{d}(\tau))}\big) = \mathrm{wrt}_i(\tau)\,. \qquad \blacktriangleleft$$

Note that Proposition 4.8.7 does not hold for all processes. Let $a \in \mathrm{Isolated}(\mathcal{A})$ be an isolated vertex in the dependence graph of $\mathcal{A}$ and let $i \in a\,\mathrm{M}$. From Lemma 4.8.4(3) we know $a\overline{a} \equiv \varepsilon$, but we have

$$\mathrm{wrt}_i(a\overline{a}) = a \neq \varepsilon = \mathrm{wrt}_i(\varepsilon) \qquad \text{and} \qquad \mathrm{rd}_i(a\overline{a}) = a \neq \varepsilon = \mathrm{rd}_i(\varepsilon)\,.$$

Now, we want to find the announced normal form of the equivalence classes of $\equiv$. To this end, we can define a (finite) trace rewriting system $\mathfrak{R}_\mathcal{A}$ (on the distributed alphabet $\mathcal{E}$) which arises from the equations in Lemma 4.8.4 by ordering them from left to right. In other words, $\mathfrak{R}_\mathcal{A}$ consists of the following rules:

(1) $[a\overline{b}] \to [\overline{b}a]$ if $a \neq b$ and $a \nparallel b$,

(2) $[a\overline{a}\overline{b}] \to [\overline{a}a\overline{b}]$ and $[ba\overline{a}] \to [b\overline{a}a]$ if $a \nparallel b$, and

(3) $[a\overline{a}] \to [\varepsilon]$ if $a \in \mathrm{Isolated}(\mathcal{A})$

for each $a, b \in A$. Note that compared to Lemma 4.8.4 in (1) there is another restriction $a \nparallel b$. However, in the case $a \parallel b$ we have $[a\overline{b}] = [\overline{b}a]$ according to our definition of $\mathcal{E}$. So, in this case the left-hand side would coincide with the right-hand side of such rule.

Next, we want to find unique, irreducible traces for each equivalence class of $\equiv$. To this end, we show the following lemma:

Lemma 4.8.8. *Let $\mathcal{A}$ be a distributed alphabet. Then the trace rewriting system $\mathfrak{R}_{\mathcal{A}}$ is terminating and confluent.*

Proof. Termination can be proved with the help of a well-founded partial ordering. To this end, let $i \in P$ be a process and $<_i$ be a linear ordering of $A_i \cup \overline{A_i}$ such that $\overline{a} <_i b$ holds for each $a, b \in A_i$. Then for each rule $\ell \to r$ of $\mathfrak{R}_{\mathcal{A}}$ there is a process $i \in P$ where $\pi_i(r)$ is length-lexicographically properly smaller than $\pi_i(\ell)$. Now, let $<$ be the product of the $<_i$'s. Then $<$ is still well-founded and we have $r < \ell$ for each rule $\ell \to r$. Hence, $\mathfrak{R}_{\mathcal{A}}$ is terminating.

Now, we show that $\mathfrak{R}_{\mathcal{A}}$ is locally confluent. The only possible overlaps of left-hand sides are the following (where $a, b, c \in A$):

- $[(b\overline{a}a)\overline{c}] \overset{(2)}{\Longleftarrow}_{\mathfrak{R}_{\mathcal{A}}} [(ba\overline{a})\overline{c}] = [b(a\overline{ac})] \overset{(2)}{\Longrightarrow}_{\mathfrak{R}_{\mathcal{A}}} [b(\overline{a}a\overline{c})]$ if $a \nparallel b, c$
- $[\overline{a}] \overset{(3)}{\Longleftarrow}_{\mathfrak{R}_{\mathcal{A}}} [(a\overline{a})\overline{a}] = [a\overline{aa}] \overset{(2)}{\Longrightarrow}_{\mathfrak{R}_{\mathcal{A}}} [\overline{a}a\overline{a}] \overset{(3)}{\Longrightarrow}_{\mathfrak{R}_{\mathcal{A}}} [\overline{a}]$ if $a \in \mathrm{Isolated}(\mathcal{A})$.

Therefore, $\mathfrak{R}_{\mathcal{A}}$ is locally confluent and, due to the termination, it is also confluent. ◂

Hence, due to termination and confluence of $\mathfrak{R}_{\mathcal{L}}$, for each action trace $\tau \in \mathbb{M}(\mathcal{E})$ there is a uniquely defined, irreducible trace $\mathrm{nf}(\tau)$ (the so-called *normal form*) with $\tau \Rightarrow^*_{\mathfrak{R}_{\mathcal{A}}} \mathrm{nf}(\tau)$. Note that we have $\tau \equiv \mathrm{nf}(\tau)$ due to the equations from Lemma 4.8.4. From the shape of the rules of $\mathfrak{R}_{\mathcal{A}}$ we learn that the set of all irreducible traces is the following trace language:

$$\mathrm{NF}_{\mathcal{Q}_{\mathcal{A}}} := \overline{\mathbb{M}(\mathcal{A})}\,\{a\overline{a} \mid a \in A \setminus \mathrm{Isolated}(\mathcal{A})\}^*\,\mathbb{M}(\mathcal{A})\,.$$

Note that isolated vertices are excluded from the "shuffled" part of the normal form since $a\overline{a} \equiv \varepsilon$ holds for $a \in \mathrm{Isolated}(\mathcal{A})$ according to Lemma 4.8.4(3).

In other words, for $\tau \in \mathbb{M}(\mathcal{E})$ there are traces $\sigma_1, \sigma_2 \in \mathbb{M}(\mathcal{A})$ and letters $a_1, a_2, \ldots, a_n \in A \setminus \mathrm{Isolated}(\mathcal{A})$ with

$$\mathrm{nf}(\tau) = \overline{\sigma_1}\, a_1\overline{a_1}a_2\overline{a_2} \ldots a_n\overline{a_n}\,\sigma_2\,.$$

For a better readability we will also write

$$\langle a_1 a_2 \ldots a_n \rangle := a_1\overline{a_1}a_2\overline{a_2} \ldots a_n\overline{a_n}\,.$$

Note that the map $\langle . \rangle \colon \mathbb{M}(\mathcal{A}) \to \mathbb{M}(\mathcal{E})$ is well-defined: let $a, b \in A$ be distinct letters with $a \parallel b$. Then we have

$$\langle ab \rangle = a\overline{a}b\overline{b} = ab\overline{a}\,\overline{b} = ba\overline{b}\,\overline{a} = b\overline{b}a\overline{a} = \langle ba \rangle\,.$$

Now, we are able to prove that there is a unique normal form for any equivalence class of the behavioral equivalence. This implies that $\equiv$ is the least congruence on $\mathbb{M}(\mathcal{E})$ satisfying the equations from Lemma 4.8.4.

Theorem 4.8.9. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\sigma, \tau \in \mathbb{M}(\mathcal{E})$ be two traces of actions. Then we have*

$$\sigma \equiv \tau \iff \mathrm{nf}(\sigma) = \mathrm{nf}(\tau)\,.$$

Proof. The implication "⇐" is obvious since

$$\sigma \equiv \mathrm{nf}(\sigma) = \mathrm{nf}(\tau) \equiv \tau$$

holds. Now, we prove the converse implication. To this end, let $\sigma_1, \sigma_2, \sigma_3, \tau_1, \tau_2, \tau_3 \in \mathbb{M}(\mathcal{A})$ with $\mathrm{nf}(\sigma) = \overline{\sigma_1}\langle\sigma_2\rangle\sigma_3$ and $\mathrm{nf}(\tau) = \overline{\tau_1}\langle\tau_2\rangle\tau_3$. Note that $\mathrm{Alph}(\sigma_2)$ and $\mathrm{Alph}(\tau_2)$ contain no letter from $\mathrm{Isolated}(\mathcal{A})$. Towards a contradiction, we assume $\sigma_1 \neq \tau_1$. Then τ_1 is not a prefix of σ_1 or vice versa. By symmetry it suffices to consider only the first case. Then we observe

$$[\![\overline{\sigma_1}\langle\sigma_2\rangle\sigma_3]\!](\sigma_1) = [\![\langle\sigma_2\rangle\sigma_3]\!](\varepsilon) = [\![\sigma_3]\!](\varepsilon) = \sigma_3 \neq \bot$$

and

$$\begin{aligned}[\![\overline{\tau_1}\langle\tau_2\rangle\tau_3]\!](\sigma_1) &= [\![\langle\tau_2\rangle\tau_3]\!]([\![\overline{\tau_1}]\!](\sigma_1)) \\ &= [\![\langle\tau_2\rangle\tau_3]\!](\bot) = \bot\,. \qquad \text{(since } \tau_1 \text{ is no prefix of } \sigma_1\text{)}\end{aligned}$$

This implies $\sigma \equiv \overline{\sigma_1}\langle\sigma_2\rangle\sigma_3 \not\equiv \overline{\tau_1}\langle\tau_2\rangle\tau_3 \equiv \tau$ which is a contradiction to our assumption $\sigma \equiv \tau$. Hence, we have $\sigma_1 = \tau_1$.

By Proposition 4.8.7 we have

$$\pi_i(\sigma_1\sigma_2) = \mathrm{rd}_i(\sigma) = \mathrm{rd}_i(\tau) = \pi_i(\tau_1\tau_2) = \pi_i(\sigma_1\tau_2)$$

for each $i \in P$ with $A_i \notin \mathrm{Isolated}(\mathcal{A})$ implying $\pi_i(\sigma_2) = \pi_i(\tau_2)$. Additionally, we have $\pi_i(\sigma_2) = \varepsilon = \pi_i(\tau_2)$ for each $i \in P$ with $A_i \subseteq \mathrm{Isolated}(\mathcal{A})$. Then from the projection-lemma (Theorem 3.4.4) we learn $\sigma_2 = \tau_2$.

Finally, we have to show $\sigma_3 = \tau_3$. Due to the definition of the duality map and due to Corollary 4.8.3 we have

$$\overline{\sigma_3{}^{\mathrm{R}}}\langle\sigma_2{}^{\mathrm{R}}\rangle\sigma_1{}^{\mathrm{R}} = \mathrm{d}(\mathrm{nf}(\sigma)) \equiv \mathrm{d}(\sigma) \equiv \mathrm{d}(\tau) \equiv \mathrm{d}(\mathrm{nf}(\tau)) = \overline{\tau_3{}^{\mathrm{R}}}\langle\tau_2{}^{\mathrm{R}}\rangle\tau_1{}^{\mathrm{R}}\,.$$

Since $\mathrm{d}(\mathrm{nf}(\sigma)), \mathrm{d}(\mathrm{nf}(\tau)) \in \mathrm{NF}_{\mathcal{Q}_\mathcal{A}}$ holds, the proof from above yields $\sigma_3{}^{\mathrm{R}} = \tau_3{}^{\mathrm{R}}$ implying $\sigma_3 = \tau_3$. ◂

4.8.3 Composition

Next we consider the composition of two action sequences in terms of their normal form. Similar to the partially lossy case, we only consider the special case $\sigma\overline{\tau}$ for traces $\sigma, \tau \in \mathbb{M}(\mathcal{A})$. The normal form of this trace is very close to the result from Lemma 4.6.6 (resp. Lemma 5.2 from [HKZ17]) in which we considered the normal form of $w\overline{r}$ where $w, r \in A^*$ in case of a single reliable queue.

In this lemma we have seen that $\mathrm{nf}_{\mathcal{Q}_\mathcal{A}}(w\overline{r}) = \overline{x}\langle y\rangle z$ holds if y is the longest word satisfying $r = xy$ and $w = yz$. In the distributed case this is very similar: the shuffled infix $\langle\rho_2\rangle$ of $\mathrm{nf}(\sigma\overline{\tau}) = \overline{\rho_1}\langle\rho_2\rangle\rho_3$ arises from the longest trace ρ_2' satisfying $\tau = \rho_1\rho_2'$ and $\sigma = \rho_2'\rho_3$ by removing isolated vertices (recall that $a\overline{a} \equiv \varepsilon$ holds for such isolated vertices $a \in \mathrm{Isolated}(\mathcal{A})$ by Lemma 4.8.4(3)).

Lemma 4.8.10. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\sigma, \tau \in \mathbb{M}(\mathcal{A})$. Let $\rho_2 \in \mathbb{M}(\mathcal{A})$ be the longest suffix of τ which also is a prefix of σ. Let $\rho_1, \rho_3 \in \mathbb{M}(\mathcal{A})$ with $\tau = \rho_1\rho_2$ and $\sigma = \rho_2\rho_3$ be the complementary prefix and suffix. Then we have $\mathrm{nf}(\sigma\overline{\tau}) \equiv \overline{\rho_1}\langle\rho_2\rangle\rho_3$. In particular, we have $\mathrm{nf}(\sigma\overline{\tau}) = \overline{\rho_1}\langle\pi_{A\setminus\mathrm{Isolated}(\mathcal{A})}(\rho_2)\rangle\rho_3$.*

Proof. Let $\mu_1, \mu_2, \mu_3 \in \mathbb{M}(\mathcal{A})$ be the uniquely defined traces with $\mathrm{nf}(\sigma\overline{\tau}) = \overline{\mu_1}\langle\mu_2\rangle\mu_3$. By Proposition 4.8.7 we know for each process $i \in P$ with $A_i \notin \mathrm{Isolated}(\mathcal{A})$:

$$\pi_i(\sigma) = \mathrm{wrt}_i(\sigma\overline{\tau}) = \mathrm{wrt}_i(\overline{\mu_1}\langle\mu_2\rangle\mu_3) = \pi_i(\mu_2\mu_3)$$

and, similarly, $\pi_i(\tau) = \pi_i(\mu_1\mu_2)$. Additionally, using Lemma 4.8.4(3) we can insert isolated vertices into μ_2 resulting in a trace $\mu_2' \in \mathbb{M}(\mathcal{A})$ with $\pi_{A \setminus \mathrm{Isolated}(\mathcal{A})}(\mu_2') = \mu_2$, $\pi_i(\sigma) = \pi_i(\mu_2'\mu_3)$ and $\pi_i(\tau) = \pi_i(\mu_1\mu_2')$ for all processes $i \in P$ (including those processes associated to isolated vertices). By the projection-lemma (Theorem 3.4.4) we learn $\sigma = \mu_2'\mu_3$ and $\tau = \mu_1\mu_2'$. Note that we have $\mathrm{nf}(\sigma\overline{\tau}) = \overline{\mu_1}\langle\mu_2\rangle\mu_3 \equiv \overline{\mu_1}\langle\mu_2'\rangle\mu_3$.

Now, let $\rho_1, \rho_2, \rho_3 \in \mathbb{M}(\mathcal{A})$ be arbitrary with $\tau = \rho_1\rho_2$ and $\sigma = \rho_2\rho_3$. Then we obtain the following:

$$\begin{aligned}
\bot \neq \rho_3 &= [\![\overline{\rho_2}]\!](\rho_2\rho_3) = [\![\overline{\rho_2}]\!](\sigma) && \text{(since } \sigma = \rho_2\rho_3\text{)}\\
&= [\![\overline{\rho_1\rho_2}]\!](\rho_1\sigma) = [\![\overline{\tau}]\!](\rho_1\sigma) && \text{(since } \tau = \rho_1\rho_2\text{)}\\
&= [\![\sigma\overline{\tau}]\!](\rho_1) = [\![\mathrm{nf}(\sigma\overline{\tau})]\!](\rho_1) && \text{(since } \sigma\overline{\tau} \equiv \mathrm{nf}(\sigma\overline{\tau})\text{)}\\
&= [\![\overline{\mu_1}\langle\mu_2'\rangle\mu_3]\!](\rho_1)\,. && \text{(since } \mathrm{nf}(\sigma\overline{\tau}) \equiv \overline{\mu_1}\langle\mu_2'\rangle\mu_3\text{)}
\end{aligned}$$

In particular, we have $[\![\overline{\mu_1}]\!](\rho_1) \neq \bot$ implying that μ_1 is a prefix of ρ_1. Since we have $\mu_1\mu_2' = \tau = \rho_1\rho_2$, Levi's lemma for traces (Theorem 3.4.6) implies that ρ_2 is a suffix of μ_2'. Since $\rho_1, \rho_2, \rho_3 \in \mathbb{M}(\mathcal{A})$ were arbitrary with $\tau = \rho_1\rho_2$ and $\sigma = \rho_2\rho_3$, μ_2' is the longest suffix of τ which is a prefix of σ. ◂

Finally, with this characterization of the composition of two simple traces we can see that each trace $\tau \in \mathbb{M}(\mathcal{E})$ has an equivalently behaving trace σ which is somewhat simple:

Corollary 4.8.11. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\tau \in \mathbb{M}(\mathcal{E})$. Then there are traces $\sigma_1, \sigma_2, \sigma_3 \in \mathbb{M}(\mathcal{A})$ with $\tau \equiv \overline{\sigma_1}\sigma_2\overline{\sigma_3}$. These traces can be computed from τ in polynomial time.*

Proof. By Theorem 4.8.9 there are uniquely defined traces $\sigma_1, \sigma_2, \sigma_3 \in \mathbb{M}(\mathcal{A})$ such that $\mathrm{nf}(\tau) = \overline{\sigma_1}\langle\sigma_2\rangle\sigma_3$. Then we learn the following:

$$\begin{aligned}
\tau &\equiv \overline{\sigma_1}\langle\sigma_2\rangle\sigma_3 && \text{(since } \tau \equiv \mathrm{nf}(\tau) = \overline{\sigma_1}\langle\sigma_2\rangle\sigma_3\text{)}\\
&= \overline{\sigma_1} \cdot (\langle\pi_{A \setminus \mathrm{Isolated}(\mathcal{A})}(\sigma_2)\rangle\sigma_3) && \text{(since } \pi_{\mathrm{Isolated}(\mathcal{A})}(\sigma_2) = \varepsilon\text{)}\\
&= \overline{\sigma_1} \cdot \mathrm{nf}(\sigma_2\sigma_3\overline{\sigma_2}) && \text{(by Lemma 4.8.10)}\\
&\equiv \overline{\sigma_1}\sigma_2\sigma_3\overline{\sigma_2}\,. && \text{(since } \mathrm{nf}(\sigma_2\sigma_3\overline{\sigma_2}) \equiv \sigma_2\sigma_3\overline{\sigma_2}\text{)}
\end{aligned}$$

We obtain $\mathrm{nf}(\tau)$ with the help of the rewriting system $\mathfrak{R}_{\mathcal{A}}$. The number of rules in this system is polynomial in $\mathcal{A}$. Since any rule transposes letters a and $\overline{b}$ or removes infixes of the form $a\overline{a}$, we can apply the rules from $\mathfrak{R}_{\mathcal{A}}$ at most polynomial many times. ◂

We can also find traces $\rho_1, \rho_2, \rho_3 \in \mathbb{M}(\mathcal{A})$ such that $\tau \equiv \rho_1\overline{\rho_2}\rho_3$: by Corollary 4.8.11 there are traces $\sigma_1, \sigma_2, \sigma_3 \in \mathbb{M}(\mathcal{A})$ with $\mathrm{d}(\tau) \equiv \overline{\sigma_1}\sigma_2\overline{\sigma_3}$. Then by duality (Corollary 4.8.3) we infer $\tau \equiv {\sigma_3}^{\mathrm{R}}\overline{{\sigma_2}^{\mathrm{R}}}{\sigma_1}^{\mathrm{R}}$.

Later, we will see that Corollary 4.8.11 is a very helpful tool to prove results concerning the reachability problem of distributed queue automata.

4.9 Conclusion

We have considered the behavioral equivalence of partially lossy queues and stacks as well as of distributed stacks and queues. In all of these cases we have found a special normal form representing the equivalence class. We have seen that the normal forms stem from the regular languages

$$\mathrm{NF}_{\mathcal{Q}_\mathcal{L}} = \overline{A}^* \left\{ wa\overline{a} \mid a \in A, w \in (F \setminus \{a\})^* \right\}^* A^* \quad \text{and} \quad \mathrm{NF}_{\mathcal{P}_\mathcal{L}} = \overline{A}^* A^* \cup \{\perp\!\!\!\perp\}$$

and from the trace languages

$$\mathrm{NF}_{\mathcal{Q}_\mathcal{A}} = \overline{\mathbb{M}(\mathcal{A})} \left\{ [a\overline{a}] \mid a \in A \setminus \mathrm{Isolated}(\mathcal{A}) \right\}^* \mathbb{M}(\mathcal{A}) \quad \text{and} \quad \mathrm{NF}_{\mathcal{P}_\mathcal{A}} = \overline{\mathbb{M}(\mathcal{A})}\mathbb{M}(\mathcal{A}) \cup \{\perp\!\!\!\perp\}\,.$$

However, for any kind of queues and any action sequence t we have also found an even simpler equivalently behaving sequence. Concretely, there are words (resp. traces) s_1, s_2, s_3 such that $t \equiv \overline{s_1} s_2 \overline{s_3}$. This fact will help us to prove several results across this thesis.

Besides these simple equivalently behaving words or traces we have also studied algebraic properties like cancellation or Green's relations. For example, we found out that none of the considered transformation monoids is cancellative. However, all of these monoids are cancellative under certain circumstances. Additionally, while Green's relations are trivial in the plq monoid there are strong connections of these relations to the aforementioned normal forms in the pls monoid. We also gave characterizations of the normal forms of compositions of two transformations. Finally, we recalled the main result from [KKP18] characterizing the embeddings into the partially lossy queue monoid. A complete picture of the relations between the plq monoids can be found in Figure 4.2.

Chapter 5
Rational Languages

5.1 Introduction

In this chapter (and the following ones) we consider special subsets of the transformation monoid of several data types. Concretely, we will study two well-known generalizations of regular languages to arbitrary monoids: so-called *rational* and *recognizable* languages. While the rational languages of a monoid are a generalization of regular expressions and the languages accepted by nondeterministic finite automata, the recognizable languages generalize the languages accepted by deterministic finite automata. For example, the rational languages of the transformation monoid of a data type $\mathcal{D}$ correspond to the control components of the $\mathcal{D}$-automata. In this connection the control component of a $\mathcal{D}$-automaton $\mathfrak{A}$ is an (extended) NFA consisting of the control states of $\mathfrak{A}$ and transitions labeled with the action sequences the automaton $\mathfrak{A}$ is allowed to apply to its memory. In other words, the control component is the automaton $\mathfrak{A}$ without its input tape.

In text books on algebraic automata theory one can find a rich theory on these two classes of monoid languages. For example, we refer to [Ber79, Pin10]. One famous result by Kleene is the coincidence of the classes of rational and recognizable languages in the free monoid [Kle51]. However, this equivalence does not hold in arbitrary monoids. For example, we will see later that the classes of rational and recognizable languages in the partially lossy queue monoid do not coincide. But at least in finitely generated monoids (like the transformation monoid of any data type) each recognizable language also is rational by McKnight's Theorem [McK64] (but not necessarily vice versa).

Here and now, we want to study the algorithmic properties of the rational languages of the partially lossy queue and stack monoids. Such properties encountered increased attention in recent years. For example, [Loh13] provides a survey on the membership problem for rational group languages.

Towards the algorithmic properties of rational languages in the pls monoid we generalize a result by Render and Kambites [RK09] to partially lossy stacks. In that paper the authors state that the rational languages in polycyclic monoids (recall that such monoids are the transformation monoids of reliable stacks) are exactly the homomorphic images of some very simple regular languages. From this result we will learn that membership, intersection emptiness, universality, and recognizability are decidable. We will also infer that the class of rational pls languages is closed under Boolean operations.

Additionally, we will see that the membership problem of rational languages in the plq monoid is decidable using nondeterministic logarithmic space, only. Moreover, we will learn that the other aforementioned problems inherit the undecidability from their counterparts in the direct product of $\mathbb{N}$ and a free monoid (cf. [CR86, GR86]). Note that we omit the

distributed variants of queues and stacks here since the results are rather similar to the partially lossy case.

Before diving into the algorithmic properties of rational languages in the pls and plq monoid we first need several definitions.

5.2 Definitions

Let $\mathbb{M}$ be a monoid. Subsets of $\mathbb{M}$ are also called $\mathbb{M}$-*languages*. An $\mathbb{M}$-language is called *rational* if it can be constructed from the finite $\mathbb{M}$-languages using union, concatenation, and (Kleene) iteration.

If $\mathbb{M} = A^*$ is the free monoid on the alphabet A the rational languages are exactly the regular languages. Hence, this definition generalizes the semantics of regular expressions from the free monoid to arbitrary monoids. Let $\phi\colon \mathbb{L} \to \mathbb{M}$ be a monoid homomorphism. Then if $S \subseteq \mathbb{L}$ is rational in $\mathbb{L}$ the $\mathbb{M}$-language $\phi(S) \subseteq \mathbb{M}$ also is rational. If ϕ is surjective and $T \subseteq \mathbb{M}$ is rational then there is a rational $\mathbb{L}$-language $S \subseteq \mathbb{L}$ with $\phi(S) = T$ [Pin10].

Remark 5.2.1. Let $\mathcal{D} = (U, \Sigma, \Theta)$ be a data type and $\mathfrak{A} = (Q, \Gamma, \mathcal{D}, I, c, \Delta, F)$ be a $\mathcal{D}$-automaton. Since we focus the transformations the automaton $\mathfrak{A}$ tries to apply to its memory in this chapter, we will remove the input tape from $\mathfrak{A}$. This results in an (extended) NFA $\mathfrak{B} = (Q, \Sigma, I, \Delta', F)$ where

$$\Delta' = \{(p, t, q) \mid \exists a \in \Gamma \cup \{\varepsilon\}\colon (p, a, t, q) \in \Delta\}$$

(recall that extended NFAs are labeled with words instead of single letters). The accepted language $L(\mathfrak{B}) \subseteq \Sigma^*$ is regular. Since $\eta_{\mathcal{D}}$ is an epimorphism (and, hence, a homomorphism), the $\mathbb{T}(\mathcal{D})$-language $\eta_{\mathcal{D}}(L(\mathfrak{B})) \subseteq \mathbb{T}(\mathcal{D})$ is rational. ⌟

There is also another generalization of regularity to arbitrary monoids using DFAs. These are the so-called recognizable languages:

Definition 5.2.2. Let $\mathbb{M}$ be a monoid. An $\mathbb{M}$-language $S \subseteq \mathbb{M}$ is called *recognizable* if there is a finite monoid $\mathbb{F}$ and a homomorphism $\phi\colon \mathbb{M} \to \mathbb{F}$ such that $S = \phi^{-1}(\phi(S))$ holds. In this case, we say that S is *recognized* by $\mathbb{F}$ (via ϕ). ⌟

Let $\mathbb{M}$ be a monoid and $S \subseteq \mathbb{M}$ be an $\mathbb{M}$-language. The *syntactic monoid* of S is $\mathbb{S}(S) := \mathbb{M}/_{\sim}$ where $\sim$ is the following congruence on $\mathbb{M}$: for two elements $m, n \in \mathbb{M}$ we have $m \sim n$ if, and only if,

$$\ell m r \in S \iff \ell n r \in S \quad \text{for all } \ell, r \in \mathbb{M}.$$

Then S is recognizable in $\mathbb{M}$ if, and only if, $\mathbb{S}(S)$ is finite. In particular, a recognizable $\mathbb{M}$-language S is recognized by its syntactic monoid $\mathbb{S}(S)$ via the natural epimorphism $\phi\colon m \mapsto [m]_{\sim}$.

In this case, we can see $\mathbb{S}(S)$ as a (deterministic) finite automaton accepting S. In this connection the set of states is $\mathbb{S}(S)$, the initial state is $\phi(e)$ (where e is the identity of $\mathbb{M}$), and the accepting states are $\phi(S)$. We have a transition from p to q labeled with $m \in \mathbb{M}$ if, and only if, $p \cdot \phi(m) = q$ holds in $\mathbb{S}(S)$. Then the accepted language of this automaton is S.

Let $\mathbb{M}$ be a monoid. Then the class of recognizable $\mathbb{M}$-languages is closed under Boolean operations and quotients (cf., e.g., [Pin10]). If $\phi\colon \mathbb{L} \to \mathbb{M}$ is a monoid homomorphism and $S \subseteq \mathbb{M}$ is recognizable in $\mathbb{M}$ then its pre-image $\phi^{-1}(S)$ is recognizable in $\mathbb{L}$ as well. Moreover,

if ϕ is surjective then also the converse implication holds, i.e., if $\phi^{-1}(S)$ is recognizable so is S [Pin10].

If $\mathbb{M}$ is a finitely generated monoid then each recognizable $\mathbb{M}$-language in $\mathbb{M}$ also is rational [McK64]. For example, this holds for the transformation monoid of any data type $\mathcal{D}$ since $\mathbb{T}(\mathcal{D})$ is generated by the finite set $\eta_{\mathcal{D}}(\Sigma)$. However, we will see in this chapter that the classes of rational and recognizable languages in the plq and pls monoid do not coincide. Nevertheless, in the free monoid A^* the classes of recognizable and rational languages coincide with the class of regular languages [Kle51].

We can also consider further subclasses of the rational and recognizable languages of a monoid:

Definition 5.2.3. Let $\mathbb{M}$ be a monoid.

(1) An $\mathbb{M}$-language $S \subseteq \mathbb{M}$ is *star-free* if it can be constructed from the finite $\mathbb{M}$-languages using union, concatenation, and complementation, only.

(2) A recognizable $\mathbb{M}$-language $S \subseteq \mathbb{M}$ is *aperiodic* if there is $n \in \mathbb{N}$ such that for each $x, y, z \in \mathbb{M}$ we have

$$xy^n z \in S \iff xy^{n+1}z \in S\,.$$

⌟

A finite monoid $\mathbb{F}$ is called *aperiodic* if there is a number $n \in \mathbb{N}$ such that for each $x \in \mathbb{F}$ we have $x^n = x^{n+1}$. In other words, $\mathbb{F}$ is aperiodic if it contains no non-trivial groups. We can see that an $\mathbb{M}$-language $S \subseteq \mathbb{M}$ is aperiodic if, and only if, its syntactic monoid $\mathbb{S}(S)$ is aperiodic.

In the free monoid A^* the classes of aperiodic and star-free languages coincide by Schützenberger's Theorem [Sch65]. By [MP71] these are exactly those languages which are accepted by so-called *counter-free* NFAs.

We can also see that the class of star-free languages is closed under Boolean operations and homomorphic images while the class of aperiodic languages is closed under Boolean operations and homomorphic preimages.

5.3 Partially Lossy Stacks

First, we consider the languages in the partially lossy stack monoid. We also call these languages *partially lossy stack languages* (or *pls languages*). One aim of this section is to characterize the rational pls languages in terms of the normal forms of action sequences which we have defined in the previous chapter. We also show that the class of rational partially lossy stack languages is a Boolean algebra. Finally, we will see that several decision problems on rational pls languages, like membership, intersection emptiness, or equality are decidable. To this end, we follow the proof strategy from Render and Kambites in [RK09] which have considered those properties for reliable stacks.

We start with a generalization of the characterization of the rational stack languages from [RK09, Theorem 5.2]. From that paper we know that a (reliable) stack language $T \subseteq \mathbb{T}(\mathcal{P}_A)$ is rational if, and only if, the normal forms $\mathrm{nf}(T)$ of the transformations in T form a regular language. We will see now that we are in the same situation for arbitrary lossiness alphabets. Note that the "if"-implication of this equivalence holds due to the preservation of rationality under homomorphisms (and the natural epimorphism η is such homomorphism mapping Σ^* to $\mathbb{T}(\mathcal{P}_\mathcal{L}) \cong \mathrm{NF}_{\mathcal{P}_\mathcal{L}}$). The converse implication is the following theorem:

Theorem 5.3.1. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $T \subseteq \mathbb{T}(\mathcal{P}_\mathcal{L})$ be a rational language. Then there is an effectively regular language $L \subseteq \mathrm{NF}_{\mathcal{P}_\mathcal{L}}$ with $\eta(L) = T$. The construction of an NFA accepting this language L is possible in polynomial time.*

Proof. Since T is rational there is a regular language $K \subseteq \Sigma^*$ with $\eta(K) = T$. Recall that $\mathfrak{S}_\mathcal{L}$ is a monadic semi-Thue system. Then by [BJW82] the set $D(K) := \{w \in \Sigma^* \mid \exists v \in K\colon v \Rightarrow^*_{\mathfrak{S}_\mathcal{L}} w\}$ of all descendants of words from K is an effectively regular set. Since $\mathrm{NF}_{\mathcal{P}_\mathcal{L}} = \overline{A}^* A^* \cup \{\perp\!\!\!\perp\}$ is regular, the set $L := D(K) \cap \mathrm{NF}_{\mathcal{P}_\mathcal{L}}$ is effectively regular as well.

Now, we have to prove that $\eta(L) = T$ holds. Since the transitions of $\mathfrak{S}_\mathcal{L}$ preserve behavioral equivalence by Lemma 4.4.1 we have

$$\eta(L) \subseteq \eta(D(K)) = \eta(K) = T\,.$$

Conversely, let $t \in T \subseteq \mathbb{T}(\mathcal{P}_\mathcal{L})$ be a transformation. Then there is $s \in K$ with $\eta(s) = t$. By the definition of $\mathfrak{S}_\mathcal{L}$ and Theorem 4.4.2 we have $\mathrm{nf}(s) \in D(K)$. Since $\mathrm{nf}(s) \in \mathrm{NF}_{\mathcal{P}_\mathcal{L}}$ we also have $\mathrm{nf}(s) \in L$ and, hence, $t = \eta(s) = \eta(\mathrm{nf}(s)) \in \eta(L)$. ◂

From Theorem 5.3.1 we infer that the class of rational pls languages is a Boolean algebra. Note that this statement generalizes Corollary 5.4 from [RK09].

Corollary 5.3.2. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet. The class of rational $\mathbb{T}(\mathcal{P}_\mathcal{L})$-languages is effectively closed under union, intersection, and complement.*

Proof. The closure under union is obvious due to the definition of rational languages. Intersection can be described in terms of union and complement. Hence, it suffices to consider the complement operation. To this end, let $T \subseteq \mathbb{T}(\mathcal{P}_\mathcal{L})$ be a rational language. Then by Theorem 5.3.1 there is an effectively regular language $L \subseteq \mathrm{NF}_{\mathcal{P}_\mathcal{L}}$ with $\eta(L) = T$. By closure properties of the class of regular languages we can compute $K := \mathrm{NF}_{\mathcal{P}_\mathcal{L}} \setminus L$. Then

$$\eta(K) = \eta(\mathrm{NF}_{\mathcal{P}_\mathcal{L}} \setminus L) = \mathbb{T}(\mathcal{P}_\mathcal{L}) \setminus \eta(L) = \mathbb{T}(\mathcal{P}_\mathcal{L}) \setminus T$$

is effectively rational as well. Note that we can compute a regular expression accepting K. Finally, from this regular expression we obtain a rational expression in $\mathbb{T}(\mathcal{P}_\mathcal{L})$ using the natural epimorphism η. ◂

Finally, we want to consider several basic decision problems having rational pls languages as their input. Concretely, we consider the following problems:

Definition 5.3.3. Let $\mathbb{M}$ be a monoid. We define the following decision problems:

(1) RATIONAL MEMBERSHIP($\mathbb{M}$): Given a rational $\mathbb{M}$-language $T \subseteq \mathbb{M}$ and an element $x \in \mathbb{M}$. Does $x \in T$ hold?

(2) RATIONAL INTERSECTION EMPTINESS($\mathbb{M}$): Given two rational $\mathbb{M}$-languages $S, T \subseteq \mathbb{M}$. Does $S \cap T = \varnothing$ hold?

(3) Rational Universality($\mathbb{M}$): Given a rational $\mathbb{M}$-language $T \subseteq \mathbb{M}$. Does $T = \mathbb{M}$ hold?

(4) Rational Inclusion($\mathbb{M}$): Given rational $\mathbb{M}$-languages $S, T \subseteq \mathbb{M}$. Does $S \subseteq T$ hold?

(5) Rational Equality($\mathbb{M}$): Given rational $\mathbb{M}$-languages $S, T \subseteq \mathbb{M}$. Does $S = T$ hold?

⌟

We should mention here, that the complexity of these problems may depend on the way we represent the rational $\mathbb{M}$-languages in the inputs of these problems, i.e., whether the languages are represented by an NFA (plus a suitable homomorphism from the free monoid into $\mathbb{M}$) or by a rational expression. However, in all considered cases in this thesis the complexities coincide for both representations.

Now, we can prove the decidability of all of these problems by reduction to their counterparts in the free monoid:

Corollary 5.3.4. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet. Then the following statements hold:*

(1) The problems Rational Membership($\mathbb{T}(\mathcal{P}_\mathcal{L})$) *and* Rational Intersection Emptiness($\mathbb{T}(\mathcal{P}_\mathcal{L})$) *are decidable in polynomial time.*

(2) The problems Rational Universality($\mathbb{T}(\mathcal{P}_\mathcal{L})$), Rational Inclusion($\mathbb{T}(\mathcal{P}_\mathcal{L})$), *and* Rational Equality($\mathbb{T}(\mathcal{P}_\mathcal{L})$) *are* PSPACE*-complete.*

Proof. All of the listed problems can be solved by polynomial time reduction to regular languages using Theorem 5.3.1. The lower complexity bound of the last three problems is inherited from their counterparts in the free monoid A^* which embeds into $\mathbb{T}(\mathcal{P}_\mathcal{L})$. ◂

Consider a unary lossiness alphabet $\mathcal{L} = (F, U)$ (i.e., we have $|F| + |U| = 1$). Then we can also show that rational membership and intersection emptiness is in NL. We can do this with the help of an on-the-fly construction of the automaton in Theorem 5.3.1 using nondeterministic logarithmic space. Since both problems are NL-complete in $A^*_\mathcal{L}$, we also have this lower complexity bound in $\mathbb{T}(\mathcal{P}_\mathcal{L})$.

If the lossiness alphabet $\mathcal{L}$ is at least binary, then both problems are also P-hard. This can be infered by reduction from the reachability problem of PDAs which is P-complete[ix].

In the next section we will also consider the problem Unique Decipherability($\mathbb{M}$) and show that it is undecidable in the plq monoid. This problem asks, whether the set T^* is a free submonoid of $\mathbb{M}$ which is generated by a given finite number of monoid elements $T \subseteq \mathbb{M}$. Until now, we do not know whether this problem is decidable in the pls monoid. However, we conjucture that it can be decided (in polynomial time?) with a variation of the Sardinas-Patterson-algorithm[x].

[ix] The reachability problem is defined later in this thesis. It is in P due to [BEM97, FWW97] (cf. Section 9.2). Its hardness is a well-known fact and can easily be obtained from the problem HornSat.

[x] The Sardinas-Patterson-algorithm solves the unique decipherability problem in the free monoid [SP53].

Additionally, we will consider the problem RATIONAL RECOGNIZABILITY($\mathbb{M}$) in the next section. This problem is defined as follows: Given a rational $\mathbb{M}$-language $T \subseteq \mathbb{M}$, is T recognizable in $\mathbb{M}$? We will see in Chapter 6 that for almost all lossiness alphabets $\mathcal{L}$ a language $T \subseteq \mathbb{T}(\mathcal{P}_\mathcal{L})$ is recognizable if, and only if, it is a trivial subset of $\mathbb{T}(\mathcal{P}_\mathcal{L})$ (i.e., if, and only if, $T = \varnothing$ or $T = \mathbb{T}(\mathcal{P}_\mathcal{L})$ holds). Hence, it is easy to solve the recognizability problem in the pls monoid.

5.4 Partially Lossy Queues[xi]

Now, we consider the languages in the plq monoid - the so-called *partially lossy queue languages* (or *plq languages*, for short). From [HKZ17, Section 8] we know that the classes of rational and recognizable queue languages do not coincide. Especially, we know that it is undecidable, whether a given rational queue language is recognizable. Additionally, [HKZ17] proved that emptiness of intersection, unique decipherability, and universality of rational queue languages are undecidable, while the rational membership problem is NL-complete in the queue monoid. Here, we will show that all of these results also hold for partially lossy queues with arbitrary underlying lossiness alphabet $\mathcal{L}$ containing at least two letters. First, we show the positive results:

5.4.1 Decidable Problems

Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$. Towards the rational membership problem we first construct an NFA accepting the equivalence class $[t]_\equiv$ (recall that $\equiv$ is the behavioral equivalence) of an action sequence $t \in \Sigma^*$. The states of this automaton represent the left-divisors of the transformation $[\![t]\!]$ in $\mathbb{T}(Q_\mathcal{L})$. Then a state representing the left-divisor $[\![s]\!]$ is reachable from the initial state (this one represents $[\![\varepsilon]\!]$) via some action sequence $s' \in \Sigma^*$ if we have $s \equiv s'$. This automaton will have $O(|t|^3)$ many states implying that $[\![t]\!]$ also has $O(|t|^3)$ many left-divisors in $\mathbb{T}(Q_\mathcal{L})$. Its construction will require at most logarithmic temporary space. Hence, we will be able to use this construction afterwards to decide the membership problem in nondeterministic logarithmic space. Note that the proof of this lemma is very close to the proof of [HKZ17, Lemma 8.1] which states this result for reliable queues, only.

Lemma 5.4.1. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$. From $t \in \Sigma^*$ we can construct an NFA accepting $[t]_\equiv$ using logarithmic space, only.*

Proof. Let $t = \alpha_1 \dots \alpha_n$ and let $0 \leq i, j, k \leq n$ be natural numbers. For the triple $q = (i, j, k)$ we define the words $q_1, q_2, q_3 \in A^*$ as follows:

- $q_1 := \mathrm{rd}(t[1, i])$ and $q_3 := \mathrm{rd}(t[i + 1, j])$ as well as
- $q_2 := \mathrm{wrt}(t[1, k])$.

We say that a triple $q = (i, j, k)$ is *good* if, and only if,

[xi] The results of this section as well as the ones of Sections 6.4 and 7.3 are published in [Köc22].

(i) there is a prefix q_2' of q_2 with $q_2' \in \mathrm{redsup}_{\mathcal{L}}(q_3)$,

(ii) $i = 0$ or $\alpha_i \in \overline{A}$ and similarly $j = 0$ or $\alpha_j \in \overline{A}$, and

(iii) $k = 0$ or $\alpha_k \in A$.

We show first, that each good triple q describes an equivalence class $[t_q]_{\equiv}$ such that $\mathrm{wrt}(t_q)$ and $\mathrm{rd}(t_q)$ are prefixes of $\mathrm{wrt}(t)$ and $\mathrm{rd}(t)$, respectively. We also show that for each $s \in \Sigma^*$ satisfying these properties there is a good triple q with $s \equiv t_q$.

▷ *Claim 1. Let $q = (i, j, k)$ be a good triple. Then there is an action sequence $t_q \in \Sigma^*$ with characteristics $\chi(t_q) = (q_2, q_1 q_3, q_3)$.*

Proof. By the definition of q_1, q_2, and q_3 we know that q_2 is a prefix of $\mathrm{wrt}(t)$ and $q_1 q_3$ is a prefix of $\mathrm{rd}(t)$. Additionally, since q is good there exists a prefix q_2' of q_2 with $q_2' \in \mathrm{redsup}_{\mathcal{L}}(q_3)$. Let q_4 be the complementary suffix of q_2 wrt. q_2'. Then $t_q := \overline{q_1}\langle\!\langle q_2', \overline{q_3}\rangle\!\rangle q_4$ satisfies $\chi(t_q) = (q_2' q_4, q_1 q_3, q_3) = (q_2, q_1 q_3, q_3)$. ◁

▷ *Claim 2. Let $s \in \Sigma^*$ where $\mathrm{wrt}(s)$ and $\mathrm{rd}(s)$ are prefixes of $\mathrm{wrt}(t)$ resp. $\mathrm{rd}(t)$. Then there is a good triple $q = (i, j, k)$ with $s \equiv t_q$.*

Proof. Recall that $\mathrm{rd}(s) = \mathrm{rd}_1(s)\mathrm{rd}_2(s)$ holds, i.e., $\mathrm{rd}(t)$ starts with $\mathrm{rd}_1(s)\mathrm{rd}_2(s)$. So, if $\mathrm{rd}_1(s) \neq \varepsilon$ there is $0 < i \leq n$ with $\mathrm{rd}(t[1, i]) = \mathrm{rd}_1(s)$ and $\alpha_i \in \overline{A}$. Otherwise we set $i := 0$. If $\mathrm{rd}_2(s) \neq \varepsilon$ there is $i < j \leq n$ with $\mathrm{rd}(t[i+1, j]) = \mathrm{rd}_2(s)$ and $\alpha_j \in \overline{A}$. Otherwise we also set $j := 0$. Finally, if $\mathrm{wrt}(s) \neq \varepsilon$ there is $1 < k \leq n$ with $\mathrm{wrt}(t[1, k]) = \mathrm{wrt}(s)$ and $\alpha_k \in A$ since $\mathrm{wrt}(s)$ is a prefix of $\mathrm{wrt}(t)$. Otherwise we set $k := 0$. Then the tuple $q := (i, j, k)$ is good: $q_2' := \mathrm{wrt}_1(s)$ is a prefix of $q_2 = \mathrm{wrt}(s)$ and $q_2' = \mathrm{wrt}_1(s) \in \mathrm{redsup}_{\mathcal{L}}(\mathrm{rd}_2(s)) = \mathrm{redsup}_{\mathcal{L}}(q_3)$. Additionally, we can see that $\chi(s) = \chi(t_q)$ holds implying $s \equiv t_q$. ◁

Now, the set of states of our NFA consists of all good triples $q = (i, j, k)$ and a unique error state $\perp\!\!\!\perp$.

The only initial state of the NFA is $\iota := (0, 0, 0)$ (this ensures $\chi(t_\iota) = (\varepsilon, \varepsilon, \varepsilon)$ and, hence, $t_\iota \equiv \varepsilon$). A state $q = (i, j, k)$ is accepting if, and only if, $t_q \equiv t$ holds (i.e., iff $\chi(t_q) = \chi(t)$).

Next, we want to define the transitions of the automaton such that, after reading of $s \in \Sigma^*$, the automaton reaches a state q with $t_q \equiv s$, provided that such state exists. Furthermore, we want to make sure that such a state exists whenever $[\![s]\!]$ is a left-divisor of $[\![t]\!]$.

So, let $q = (i, j, k)$ be a state (i.e., q is good) and $a \in A$. To define the state reached from q after writing a, let $k' > k$ be the minimal write-position in t after k. In other words, we have $k' > k$, $\alpha_{k'} \in A$, and $t[k+1, k'-1] \in \overline{A}^*$. If there is no such k' or if $\alpha_{k'} \neq a$ then the NFA ends up in the error state $\perp\!\!\!\perp$. Otherwise it moves to $p := (i, j, k')$, which is a good triple and, hence, a state of the automaton. Then we have

$$\begin{aligned}\chi(t_q \cdot a) &= (\mathrm{wrt}(t[1, k])\, a, \mathrm{rd}(t[1, j]), \mathrm{rd}(t[i+1, j])) \\ &= (\mathrm{wrt}(t[1, k']), \mathrm{rd}(t[1, j]), \mathrm{rd}(t[i+1, j])) \\ &= \chi(t_p)\,.\end{aligned}$$

Now, we define which state is reached from q after executing $\overline{a}$. Let $j' > j$ be the minimal read-position in t after j. In other words, we have $j' > j$, $\alpha_{j'} \in \overline{A}$, and $t[j+1, j'-1] \in A^*$. If there is no such j' or if $\alpha_{j'} \neq \overline{a}$, then the NFA ends up in the error state $\perp\!\!\!\perp$. So, assume that such j' exists and $\alpha_{j'} = \overline{a}$ holds. Consider the word $s = \mathrm{sws}(\mathrm{wrt}(t[1, k]), \mathrm{rd}(t[i+1, j])a)$. Then we have $\mathrm{rd}_2(t_q \overline{a}) = s$ by Lemma 4.7.14. Set $i' \geq i$ such that $i' = 0$ or $\alpha_{i'} \in \overline{A}$ holds as well as $\mathrm{rd}(t[i'+1, j']) = s$. Then $q_2' \in \mathrm{redsup}_{\mathcal{L}}(\mathrm{rd}(t[i'+1, j']))$ for a prefix q_2' of $\mathrm{wrt}(t[1, k])$ by

definition of s. Hence, $p := (i', j', k)$ is good and, therefore, a valid state of the NFA and we put an $\overline{a}$-labeled edge from q to p. We obtain

$$\begin{aligned}\chi(t_q \cdot \overline{a}) &= (\mathrm{wrt}(t[1,k]), \mathrm{rd}(t[1,j])\, a, s) && \text{(by Lemma 4.7.14)}\\ &= (\mathrm{wrt}(t[1,k]), \mathrm{rd}(t[1,j']), \mathrm{rd}(t[i'+1,j'])) \\ &= \chi(t_p)\,.\end{aligned}$$

This finishes the construction of the NFA.

Now, let $s \in \Sigma^*$. If there is an s-labeled path from $\iota = (0,0,0)$ to a non-error state q we obtain $\chi(s) = \chi(t_q)$ by induction on $|s|$ from the above calculations. In particular, if q is an accepting state of our NFA, we know $t_q \equiv t$ implying $s \in [t]_{\equiv}$.

Next, let $[\![s]\!]$ be a left-divisor of $[\![t]\!]$. Then $\mathrm{wrt}(s)$ and $\mathrm{rd}(s)$ are prefixes of $\mathrm{wrt}(t)$ and $\mathrm{rd}(t)$, respectively, since $\mathrm{wrt}, \mathrm{rd}\colon \Sigma^* \to A^*$ are homomorphisms. Then by induction on $|s|$ we obtain an s-labeled path from ι to a non-error state q with $\chi(s) = \chi(t_q)$. In particular, if $s \in [t]_{\equiv}$ then we have $\chi(t_q) = \chi(s) = \chi(t)$, i.e., q is accepting. Thus, the NFA accepts $[t]_{\equiv}$.

By the construction of the NFA, it is clear that a Turing-machine with t on its input tape can, using logarithmic space on its work tape, write the list of all transitions on its one-way output tape. ◂

Note that the NFA we have constructed in the proof of Lemma 5.4.1 is actually deterministic. However, we do not require determinism for the following statement. This theorem states that the rational membership problem is decidable in the plq monoid:

Theorem 5.4.2. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$. Then the problem* RATIONAL MEMBERSHIP$(\mathbb{T}(Q_{\mathcal{L}}))$ *is* NL*-complete. This complexity result also holds if the underlying lossiness alphabet $\mathcal{L}$ is part of the problem's input.*

Proof. Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$, $t \in \Sigma^*$, and $L \subseteq \mathbb{T}(Q_{\mathcal{L}})$ be rational. If L is given by a rational expression, we first have to construct an NFA from this expression. This is possible using only logarithmic space with the help of an on-the-fly construction: the states of the NFA are the positions in the rational expression and the transitions are defined accordingly. So, from now on let $\mathfrak{A}$ be given by an NFA over Σ. By Lemma 5.4.1 there is also an NFA $\mathfrak{B}$ accepting $[t]_{\equiv}$ which can be constructed using only logarithmic space.

Then there exists $s \in L(\mathfrak{A})$ with $s \equiv t$ if, and only if, $L(\mathfrak{A}) \cap [t]_{\equiv} \neq \varnothing$ if, and only if, $L(\mathfrak{A}) \cap L(\mathfrak{B}) \neq \varnothing$. Using an on-the-fly construction of $\mathfrak{B}$, this can be decided nondeterministically in logarithmic space. Hence, the problem is in NL.

Since the free monoid A^* embeds into $\mathbb{T}(Q_{\mathcal{L}})$ and since the rational subset membership problem for A^* is NL-hard [Jon75], we also get NL-hardness for $\mathbb{T}(Q_{\mathcal{L}})$. ◂

5.4.2 Undecidable Problems

Next we will prove some negative algorithmic results on rational languages of the plq monoid. Concretely, we will see that the remaining problems defined in the previous section are undecidable for partially lossy queues.

In [HKZ17, Section 8] these undecidabilities for reliable queues could be inferred from an embedding of $\{a,b\}^* \times \{c,d\}^*$ into $\mathbb{T}(\mathcal{Q}_{(\varnothing,A)})$ for any at least binary alphabet A. Unfortunately, this does not work in arbitrary plq monoids since this direct product does not embed into $\mathbb{T}(\mathcal{Q}_{(\{a,b\},\varnothing)})$ by Theorem 4.7.20. Though, we can prove all the undecidability results considered in [HKZ17] for any plq monoid with at least binary underlying alphabet $A = F \cup U$.

Some of our results base on an embedding of the monoid $\{a\}^* \times \{b,c\}^*$ into $\mathbb{T}(\mathcal{Q}_{\mathcal{L}})$. Unfortunately, this does not help for unique decipherability and rational intersection emptiness since their counterparts in $\{a\}^* \times \{b,c\}^*$ are decidable. Hence, we have to prove them directly.

First, we consider the problem Unique Decipherability($\mathbb{M}$) of a monoid $\mathbb{M}$, i.e., the question whether a given finite set $T \subseteq \mathbb{M}$ freely generates T^*. In this context, a set $T \subseteq \mathbb{M}$ *freely generates* T^* if for any sequences $s_1, \ldots, s_k \in T$ and $t_1, \ldots, t_\ell \in T$ the equation $s_1 \ldots s_k = t_1 \ldots t_\ell$ implies $k = \ell$ and $s_i = t_i$ for each $1 \leq i \leq k = \ell$. We also say that T^* is *uniquely decipherable* by T in this case.

We prove the undecidability of the Unique Decipherability($\mathbb{T}(\mathcal{Q}_{\mathcal{L}})$) by reduction from its counterpart in $\{a,b\}^* \times \{c,d\}^*$. We do this with the help of a special encoding of the given elements and a further special item.

Theorem 5.4.3. *Let $\mathcal{L} = (F,U)$ be a lossiness alphabet with $|F| + |U| \geq 2$. Then the problem* Unique Decipherability($\mathbb{T}(\mathcal{Q}_{\mathcal{L}})$) *is undecidable.*

Proof. We prove this theorem by reduction of Unique Decipherability($\{a,b\}^* \times \{c,d\}^*$), which is undecidable by [CR86, Theorem 3.1]. So, let $a, b \in A$ be distinct letters and

$$S := \{(x_1, y_1), \ldots, (x_k, y_k)\} \subseteq \{a,b\}^* \times \{c,d\}^* .$$

▷ *Idea.* We will now encode these tuples as transformations $t_1, \ldots, t_k$ over $\{a,b\}$ as follows: the components $x_1, \ldots, x_k$ are mapped to sequences of write actions and the components $y_1, \ldots, y_k$ are mapped to read action sequences. Unfortunately, while the x_i's and the y_j's (for $1 \leq i, j \leq n$) commute in $\{a,b\}^* \times \{c,d\}^*$, it is not necessarily the case that their encodings also commute (recall that the commutations of write and read actions depend on some special contexts). But we can simulate the full commutativity of these two components in our encodings by appending a big number of read actions (this is according to Lemma 4.7.14). This can be done by introduction of another transformation t_0 containing more read than write actions which we append to our encodings sufficiently often. ⌟

So, we first define the embeddings $f\colon \{a,b\}^* \to A^*$ and $g\colon \{c,d\}^* \to A^*$ by $f(a) = g(c) = aa$ and $f(b) = g(d) = ab$. Set $t_0 := [\![bb\overline{bbbb}]\!]$, $t_i := [\![f(x_i)\overline{g(y_i)}]\!]$ for any $1 \leq i \leq k$, and

$$T := \{t_i \mid 0 \leq i \leq k\} \subseteq \mathbb{T}(\mathcal{Q}_{\mathcal{L}}) .$$

We show now that S^* is freely generated by S if, and only if, T^* is freely generated by T.
▷ *Claim 1.* *If T^* is freely generated by T then S^* is freely generated by S.*

Proof. We prove this claim by contraposition. So, assume that S^* is not freely generated by S. Then there are two different sequences of indices $(i_1, \ldots, i_m) \neq (j_1, \ldots, j_n)$ where $m, n > 0$ such that

$$(x_{i_1} \ldots x_{i_m}, y_{i_1} \ldots y_{i_m}) = (x_{j_1} \ldots x_{j_n}, y_{j_1} \ldots y_{j_n}) .$$

Let $\ell = |f(x_{i_1} \ldots x_{i_m})|$. Set $p := t_{i_1} \ldots t_{i_m} \cdot t_0^\ell$ and $q := t_{j_1} \ldots t_{j_n} \cdot t_0^\ell$. It is a simple exercise to prove $\mathrm{nf}(t_0^\ell) = \overline{b}^\ell \langle\!\langle b^{2\ell}, \overline{b^{2\ell}} \rangle\!\rangle$. Additionally, by the choice of the two sequences of indices we have

$$\begin{aligned} \mathrm{wrt}(t_{i_1} \ldots t_{i_m}) &= f(x_{i_1} \ldots x_{i_m}) = f(x_{j_1} \ldots x_{j_n}) = \mathrm{wrt}(t_{j_1} \ldots t_{j_n}) \quad \text{and} \\ \mathrm{rd}(t_{i_1} \ldots t_{i_m}) &= g(y_{i_1} \ldots y_{i_m}) = g(y_{j_1} \ldots y_{j_n}) = \mathrm{rd}(t_{j_1} \ldots t_{j_n}) . \end{aligned} \tag{5.1}$$

Since $|\mathrm{wrt}(t_{i_1} \ldots t_{i_m})|_b < \ell$ holds (by the choice of ℓ and f) we obtain a number $0 \leq k < \ell$ with:

$$\begin{aligned} b^k &= \mathrm{sws}(\mathrm{wrt}(t_{i_1} \ldots t_{i_m}), \mathrm{rd}_2(t_{i_1} \ldots t_{i_m}) \cdot b^\ell) \\ &= \mathrm{sws}(\mathrm{wrt}(t_{i_1} \ldots t_{i_m}), b^\ell) \\ &= \mathrm{sws}(\mathrm{wrt}(t_{j_1} \ldots t_{j_n}), \mathrm{rd}_2(t_{j_1} \ldots t_{j_n}) \cdot b^\ell) . \end{aligned}$$

Finally, by application of Lemma 4.7.14 and Corollary 4.7.15 we infer

$$\begin{aligned} p &= \left[\!\left[\overline{\mathrm{rd}_1(t_{i_1} \ldots t_{i_m})} \mathrm{wrt}(t_{i_1} \ldots t_{i_m}) \overline{\mathrm{rd}_2(t_{i_1} \ldots t_{i_m})} \cdot \overline{b}^\ell \cdot \langle\!\langle b^{2\ell}, \overline{b^{2\ell}} \rangle\!\rangle \right]\!\right] && \text{(by Corollary 4.7.15)} \\ &= \left[\!\left[\overline{\mathrm{rd}_1(t_{i_1} \ldots t_{i_m}) \mathrm{rd}_2(t_{i_1} \ldots t_{i_m}) b^{\ell-k}} \mathrm{wrt}(t_{i_1} \ldots t_{i_m}) \overline{b}^k \cdot \langle\!\langle b^{2\ell}, \overline{b^{2\ell}} \rangle\!\rangle \right]\!\right] && \text{(by Lemma 4.7.14)} \\ &= \left[\!\left[\overline{\mathrm{rd}_1(t_{j_1} \ldots t_{j_n}) \mathrm{rd}_2(t_{j_1} \ldots t_{j_n}) b^{\ell-k}} \mathrm{wrt}(t_{j_1} \ldots t_{j_n}) \overline{b}^k \cdot \langle\!\langle b^{2\ell}, \overline{b^{2\ell}} \rangle\!\rangle \right]\!\right] && \text{(by Equation (5.1))} \\ &= \left[\!\left[\overline{\mathrm{rd}_1(t_{j_1} \ldots t_{j_n})} \mathrm{wrt}(t_{j_1} \ldots t_{j_n}) \overline{\mathrm{rd}_2(t_{j_1} \ldots t_{j_n})} \cdot \overline{b}^\ell \cdot \langle\!\langle b^{2\ell}, \overline{b^{2\ell}} \rangle\!\rangle \right]\!\right] && \text{(by Lemma 4.7.14)} \\ &= q . && \text{(by Corollary 4.7.15)} \end{aligned}$$

Accordingly T^* is not freely generated by T. ◁

▷ *Claim 2. If S^* is freely generated by S then T^* is freely generated by T.*

Proof. Again, we prove this implication by contraposition. Assume that T^* is not freely generated by T. If $[\![\varepsilon]\!] \in T$ and, hence, $(\varepsilon, \varepsilon) \in S$ holds, S^* is obviously not freely generated by S. So, we can assume that $[\![\varepsilon]\!] \notin T$ holds and, hence, $(\varepsilon, \varepsilon) \notin S$. Then there are indices $(i_1, \ldots, i_m) \neq (j_1, \ldots, j_n)$ with $m, n > 0$ such that $t_{i_1} \ldots t_{i_m} = t_{j_1} \ldots t_{j_n}$ holds. Let $(i'_1, \ldots, i'_{m'}), (j'_1, \ldots, j'_{n'})$ be the above sequences after deletion of all 0's. Since t_0 is the only element in T adding bb into the projections of write and read actions and due to Proposition 4.7.4, t_0 does not commute with any t_i where $1 \leq i \leq k$. Hence, we still have $(i'_1, \ldots, i'_{m'}) \neq (j'_1, \ldots, j'_{n'})$, $\mathrm{wrt}(t_{i'_1} \ldots t_{i'_{m'}}) = \mathrm{wrt}(t_{j'_1} \ldots t_{j'_{n'}})$ and $\mathrm{rd}(t_{i'_1} \ldots t_{i'_{m'}}) = \mathrm{rd}(t_{j'_1} \ldots t_{j'_{n'}})$. Then we have

$$\begin{aligned} f(x_{i'_1} \ldots x_{i'_{m'}}) &= \mathrm{wrt}(t_{i'_1} \ldots t_{i'_{m'}}) = \mathrm{wrt}(t_{j'_1} \ldots t_{j'_{n'}}) = f(x_{j'_1} \ldots x_{j'_{n'}}) , \\ g(y_{i'_1} \ldots y_{i'_{m'}}) &= \mathrm{rd}(t_{i'_1} \ldots t_{i'_{m'}}) = \mathrm{rd}(t_{j'_1} \ldots t_{j'_{n'}}) = g(y_{j'_1} \ldots y_{j'_{n'}}) . \end{aligned}$$

By injectivity of f and g we infer that

$$(x_{i'_1} \ldots x_{i'_{m'}}, y_{i'_1} \ldots y_{i'_{m'}}) = (x_{j'_1} \ldots x_{j'_{n'}}, y_{j'_1} \ldots y_{j'_{n'}})$$

holds, i.e., S^* is not freely generated by S. ◁

Finally, we have seen that S^* is freely generated by S if, and only if, T^* is freely generated by T. So, we have reduced Unique Decipherability$(\{a,b\}^* \times \{c,d\}^*)$ to Unique Decipherability$(\mathbb{T}(\mathcal{Q}_\mathcal{L}))$. Since the former problem is undecidable by [CR86], the latter one also is undecidable. ◂

The next problem to consider is the emptiness of intersections of two rational plq languages. Given two recognizable plq languages, this problem is decidable since the class of recognizable languages is effectively closed under intersection. However, we will prove that this decidability does not hold for arbitrary rational plq languages.

As a corollary we can infer that the class of rational plq languages is not effectively closed under intersection and complement. Afterwards, we will prove the existence of two rational languages whose intersection is not rational. In other words, the classes of rational and recognizable plq languages do not coincide. Nevertheless, each recognizable language in $\mathbb{T}(\mathcal{Q}_\mathcal{L})$ is rational due to [McK64] since the plq monoid is finitely generated.

Theorem 5.4.4. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$. Then the problem* Rational Intersection Emptiness$(\mathbb{T}(\mathcal{Q}_\mathcal{L}))$ *is undecidable.*

Proof. We prove this by reduction of Post's Correspondence Problem (PCP) which is undecidable by [Pos46]. So, let $a, b \in A$ be two distinct letters and $I = (x_i, y_i)_{1 \leq i \leq k}$ be an instance of the PCP with $x_i, y_i \in A^*$.

▷ *Idea.* The reduction is similar to the proof of Theorem 5.4.3. Here, for each tuple (x_i, y_i) we introduce two transformations s_i and t_i in which the write action sequences encode (unary) the index i and the read actions coincide with x_i resp. y_i. We then obtain two rational plq languages S_I and T_I consisting of non-empty sequences of s_i's resp. t_i's and arbitrarily many additional read actions. The additional read actions are necessarily to achieve the commutativity of the write and read action sequences of the s_i's and t_i's. ⌟

For $1 \leq i \leq k$ define the transformations $s_i := [\![a^i b \overline{x_i}]\!]$ and $t_i := [\![a^i b \overline{y_i}]\!]$. Then we can define rational plq languages as follows:

$$S_I := \{s_i \mid 1 \leq i \leq k\}^+ [\![\overline{ab}^*]\!] \quad \text{and} \quad T_I := \{t_i \mid 1 \leq i \leq k\}^+ [\![\overline{ab}^*]\!] .$$

Now we show that $S_I \cap T_I \neq \varnothing$ holds if, and only if, I has a solution.

▷ *Claim 1. If $S_I \cap T_I \neq \varnothing$ holds, then I has a solution.*

Proof. Let $t \in S_I \cap T_I$. Then by definition of S_I and T_I there are $\ell \in \mathbb{N}$ and indices $i_1, \ldots, i_m$ and $j_1, \ldots, j_n$ (where $m, n > 0$) such that $s_{i_1} \ldots s_{i_m} [\![\overline{ab}^\ell]\!] = t = t_{j_1} \ldots t_{j_n} [\![\overline{ab}^\ell]\!]$ holds. Then we have

$$x_{i_1} \ldots x_{i_m} ab^\ell = \mathrm{rd}(t) = y_{j_1} \ldots y_{j_n} ab^\ell$$

implying $x_{i_1} \ldots x_{i_m} = y_{j_1} \ldots y_{j_n}$. By

$$a^{i_1} b \ldots a^{i_m} b = \mathrm{wrt}(t) = a^{j_1} b \ldots a^{j_n} b$$

we can infer $(i_1, \ldots, i_m) = (j_1, \ldots, j_n)$ which is a solution of I. ◁

▷ *Claim 2. If I has a solution, then we have $S_I \cap T_I \neq \varnothing$.*

Proof. Let $(i_1, \dots, i_m)$ with $m > 0$ be a solution of I, i.e., we have $x_{i_1} \dots x_{i_m} = y_{i_1} \dots y_{i_m}$. Set $s' := s_{i_1} \dots s_{i_m}$ and $t' := t_{i_1} \dots t_{i_m}$ and set $s := s'[\![\overline{a}\overline{b}^{m+1}]\!] \in S_I$ and $t := t'[\![\overline{a}\overline{b}^{m+1}]\!] \in T_I$. Then we have $\mathrm{wrt}(s) = \mathrm{wrt}(t)$ and $\mathrm{rd}(s) = \mathrm{rd}(t)$. Furthermore, by Corollary 4.7.15 we have

$$s = [\![\overline{\mathrm{rd}_1(s')}\mathrm{wrt}(s)\overline{\mathrm{rd}_2(s')}\,\overline{ab^{m+1}}]\!] \quad \text{and} \quad t = [\![\overline{\mathrm{rd}_1(t')}\mathrm{wrt}(t)\overline{\mathrm{rd}_2(t')}\,\overline{ab^{m+1}}]\!] .$$

Note that we have $|\mathrm{wrt}(s)|_b = |\mathrm{wrt}(t)|_b = m$. Hence, there exists a number $0 \le k \le m$ with

$$\begin{aligned} b^k &= \mathrm{sws}(\mathrm{wrt}(s), \mathrm{rd}_2(s')ab^{m+1}) && (\text{since } |\mathrm{wrt}(s)|_b < m+1) \\ &= \mathrm{sws}(\mathrm{wrt}(s), b^{m+1}) = \mathrm{sws}(\mathrm{wrt}(t), b^{m+1}) && (\text{since } \mathrm{wrt}(s) = \mathrm{wrt}(t)) \\ &= \mathrm{sws}(\mathrm{wrt}(t), \mathrm{rd}_2(t')ab^{m+1}) \end{aligned}$$

implying $\mathrm{rd}_2(s) = b^k = \mathrm{rd}_2(t)$. This finally implies $\chi(s) = \chi(t)$, i.e., $s = t \in S_I \cap T_I \neq \emptyset$. ◁

Hence, we reduced PCP to RATIONAL INTERSECTION EMPTINESS($\mathbb{T}(Q_{\mathcal{L}})$) which is therefore undecidable. ◂

To prove that the rational plq languages are not closed under intersection and to prove the undecidability of the next problems we use an embedding of $\{a\}^* \times \{b, c\}^*$ into the plq monoid. Assume that $a, b \in A$ holds and that a and b are distinct letters. Such an embedding is $\psi\colon \{a\}^* \times \{b, c\}^* \hookrightarrow \mathbb{T}(Q_{\mathcal{L}})$ which is induced by $\psi(a, \varepsilon) = [\![a]\!]$, $\psi(\varepsilon, b) = [\![\overline{a}\overline{b}]\!]$ and $\psi(\varepsilon, c) = [\![\overline{a}\overline{bb}]\!]$ (cf. Theorem 4.7.20 and [KKP18, Lemma 6.17]). Note that there are some commutations in $\{a\}^* \times \{b, c\}^*$:

$$(a, \varepsilon) \cdot (\varepsilon, b) = (a, b) = (\varepsilon, b) \cdot (a, \varepsilon) \quad \text{and} \quad (a, \varepsilon) \cdot (\varepsilon, c) = (a, c) = (\varepsilon, c) \cdot (a, \varepsilon) .$$

The map ψ preserves these commutations:

$$\begin{aligned} \psi(a, \varepsilon) \cdot \psi(\varepsilon, b) = [\![a\overline{a}\overline{b}]\!] = [\![\overline{a}\overline{b}a]\!] = \psi(\varepsilon, b) \cdot \psi(a, \varepsilon) \quad \text{and} \\ \psi(a, \varepsilon) \cdot \psi(\varepsilon, c) = [\![a\overline{a}\overline{bb}]\!] = [\![\overline{a}\overline{bb}a]\!] = \psi(\varepsilon, c) \cdot \psi(a, \varepsilon) \end{aligned} \tag{5.2}$$

according to the equations from Lemma 4.7.2.

Theorem 5.4.5. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \ge 2$. The class of rational $\mathbb{T}(Q_{\mathcal{L}})$-languages is not closed under intersection. In particular, the class of recognizable $\mathbb{T}(Q_{\mathcal{L}})$-languages is a proper subclass of the rational $\mathbb{T}(Q_{\mathcal{L}})$-languages.*

Proof. Consider the following rational relations in $\{a\}^* \times \{b, c\}^*$:

$$X := \{(a^m, b^m c^n) \mid m, n \in \mathbb{N}\} \quad \text{and} \quad Y := \{(a^m, b^n c^m) \mid m, n \in \mathbb{N}\} .$$

Then $\psi(X)$ and $\psi(Y)$ are rational in $\mathbb{T}(Q_{\mathcal{L}})$. Suppose that $\psi(X) \cap \psi(Y)$ is rational. Then there is a regular language $L \subseteq \Sigma^*$ with $\eta(L) = \psi(X) \cap \psi(Y)$. Since ψ is injective we have

$$\psi(X) \cap \psi(Y) = \psi(X \cap Y) = \psi(\{(a^n, b^n c^n) \mid n \in \mathbb{N}\})$$

Hence, by definition of ψ we have $\mathrm{rd}(L) = \{(ab)^n (abb)^n \mid n \in \mathbb{N}\}$ which would be regular since rd is a homomorphism. But this is a contradiction to the Pumping Lemma.

Towards the second statement, recall that $\mathbb{T}(Q_{\mathcal{L}})$ is finitely generated. Then from [McK64] we know that each recognizable plq language is rational. From the first statement and from the fact that the class of recognizable languages is closed under intersection, we infer that $\psi(X)$ and $\psi(Y)$ are rational but not recognizable. ◂

From the previous theorem we also infer that the class of rational plq languages is not closed under complement. Otherwise closure under complement and union would also imply closure under intersection, contradicting our theorem above.

By $\mathbb{I}_{\mathcal{L}} \subseteq \mathbb{T}(\mathcal{Q}_{\mathcal{L}})$ we denote the image of $\underline{\psi}$, i.e., $\underline{\psi}$ is an isomorphism from $\{a\}^* \times \{b,c\}^*$ into $\mathbb{I}_{\mathcal{L}}$. It is easy to see that $\mathbb{I}_{\mathcal{L}} = \{[\![a]\!], [\![\overline{ab}]\!], [\![\overline{abb}]\!]\}^*$ holds. In the following lemma we prove that recognizability in this submonoid implies recognizability in the whole plq monoid.

Lemma 5.4.6. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ and $T \subseteq \mathbb{I}_{\mathcal{L}}$ be recognizable in $\mathbb{I}_{\mathcal{L}}$. Then T is recognizable in $\mathbb{T}(\mathcal{Q}_{\mathcal{L}})$.*

Proof. Let $T \subseteq \mathbb{I}_{\mathcal{L}}$ be recognizable in $\mathbb{I}_{\mathcal{L}}$. Then $\psi^{-1}(T) \subseteq \{a\}^* \times \{b,c\}^*$ is recognizable in $\{a\}^* \times \{b,c\}^*$. Due to Mezei's Theorem [Ber79, Theorem III.1.5] there are regular languages $V_i \subseteq \{a\}^*$ and $W_i \subseteq \{b,c\}^*$ with

$$\psi^{-1}(T) = \bigcup_{1 \leq i \leq k} V_i \times W_i \,.$$

Now we define two homomorphisms $g\colon \{a\}^* \to A^*$ by $g(a) = a$ and $h\colon \{b,c\}^* \to A^*$ by $h(b) = ab$ and $h(c) = abb$. Note that we have $[\![g(a)]\!] = \psi(a, \varepsilon)$, $[\![\overline{h(b)}]\!] = \psi(\varepsilon, b)$, and $[\![\overline{h(c)}]\!] = \psi(\varepsilon, c)$. Then $g(V_i), h(W_i) \subseteq A^*$ are regular as well. Hence, $\mathrm{wrt}^{-1}(g(V_i))$ and $\mathrm{rd}^{-1}(h(W_i))$ are recognizable in $\mathbb{T}(\mathcal{Q}_{\mathcal{L}})$ and therefore

$$\bigcup_{1 \leq i \leq k} \mathrm{wrt}^{-1}(g(V_i)) \cap \mathrm{rd}^{-1}(h(W_i))$$

also is recognizable. Finally we have to prove that this language equals T:

▷ *Claim.* *We have $T = \bigcup_{1 \leq i \leq k} \mathrm{wrt}^{-1}(g(V_i)) \cap \mathrm{rd}^{-1}(h(W_i))$.*

Proof. First, let $t \in T \subseteq \mathbb{I}_{\mathcal{L}}$. Then there is a tuple $(v, w) \in \{a\}^* \times \{b,c\}^*$ with $\psi(v, w) = t$. Since $(v, w) \in \psi^{-1}(t) \subseteq \psi^{-1}(T)$ there is $1 \leq i \leq k$ with $v \in V_i$ and $w \in W_i$. Then we have

$$\begin{aligned} t = \psi(v, w) &= \psi(v, \varepsilon)\psi(\varepsilon, w) = [\![g(v)\overline{h(w)}]\!] && \text{(by Equation (5.2))} \\ &\in \mathrm{wrt}^{-1}(g(v)) \cap \mathrm{rd}^{-1}(h(w)) \subseteq \mathrm{wrt}^{-1}(g(V_i)) \cap \mathrm{rd}^{-1}(h(W_i)) \,. \end{aligned}$$

Conversely, let $1 \leq i \leq k$ and $t \in \mathrm{wrt}^{-1}(g(V_i)) \cap \mathrm{rd}^{-1}(h(W_i))$. Then there are $v \in V_i$ and $w \in W_i$ with $\mathrm{wrt}(t) = g(v)$ and $\mathrm{rd}(t) = h(w)$. From the definition of g and h we infer $\mathrm{wrt}(t) \in a^*$ and $\mathrm{rd}(t) \in \{\varepsilon\} \cup A^*b$. Since $a \neq b$ holds, we learn that ε is the only suffix r_2 of $\mathrm{rd}(t)$ having a prefix w_1 of $\mathrm{wrt}(t)$, which is a reduced $\mathcal{L}$-superword of r_2. Hence, we have $\mathrm{rd}_2(t) = \varepsilon$ and therefore $t = [\![\overline{h(w)}g(v)]\!] = \psi(v, w)$. Finally, from $v \in V_i$ and $w \in W_i$ we learn $(v, w) \in \psi^{-1}(T)$ implying $t = \psi(v, w) \in T$. ◁

All in all, we have seen that T is recognizable in $\mathbb{T}(\mathcal{Q}_{\mathcal{L}})$. ◂

To prove the undecidability of the rational universality and the recognizability problem we use the embedding ψ and the results from Gibbons and Rytter [GR86] which state that the counterparts of these problems in $\{a\}^* \times \{b,c\}^*$ are undecidable. Note that rational universality can be reduced to rational inclusion and equality. Hence, these two problems also are undecidable in the plq monoid.

Theorem 5.4.7. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$. Then the following statements hold:*

(1) The problem RATIONAL UNIVERSALITY$(\mathbb{T}(Q_{\mathcal{L}}))$ *is undecidable. Consequently, the problems* RATIONAL INCLUSION$(\mathbb{T}(Q_{\mathcal{L}}))$ *and* RATIONAL EQUALITY$(\mathbb{T}(Q_{\mathcal{L}}))$ *also are undecidable.*

(2) The problem RATIONAL RECOGNIZABILITY$(\mathbb{T}(Q_{\mathcal{L}}))$ *is undecidable.*

Proof. We first prove (1). To this end, let $T \subseteq \{a\}^* \times \{b, c\}^*$ be rational. Then $\psi(T)$ is rational in $\mathbb{T}(Q_{\mathcal{L}})$. Due to Lemma 5.4.6 the language $\mathbb{I}_{\mathcal{L}}$ is recognizable in $\mathbb{T}(Q_{\mathcal{L}})$. Therefore $\mathbb{T}(Q_{\mathcal{L}}) \setminus \mathbb{I}_{\mathcal{L}}$ is recognizable and, hence, rational in $\mathbb{T}(Q_{\mathcal{L}})$ since this monoid is finitely generated. Consequently, $\psi(T) \cup (\mathbb{T}(Q_{\mathcal{L}}) \setminus \mathbb{I}_{\mathcal{L}})$ is rational as well. This plq language equals $\mathbb{T}(Q_{\mathcal{L}})$ if, and only if, $\psi(T) = \mathbb{I}_{\mathcal{L}}$, i.e., $T = \{a\}^* \times \{b, c\}^*$. But this latter question is undecidable by [GR86, Theorem 2(Q4)].

Finally, we prove (2). Let $T \subseteq \{a\}^* \times \{b, c\}^*$ be rational. Then $\psi(T)$ is rational in $\mathbb{T}(Q_{\mathcal{L}})$. By Lemma 5.4.6 $\psi(T)$ is recognizable in $\mathbb{T}(Q_{\mathcal{L}})$ if, and only if, it is recognizable in $\mathbb{I}_{\mathcal{L}}$. This is the case if, and only if, T is recognizable in $\{a\}^* \times \{b, c\}^*$. But this latter question is undecidable by [GR86, Theorem 2(Q6)]. ◂

This theorem states that it is undecidable whether $\eta(L)$ is recognizable for a given regular language $L \subseteq \Sigma^*$. This question is equivalent to ask whether the language $\eta^{-1}(\eta(L))$ is regular if $L \subseteq \Sigma^*$ is a regular language. However, we are able to decide a related problem in the free monoid of basic queue actions. Concretely, we can decide whether, for a given regular language $L \subseteq \Sigma^*$, the equation $\eta^{-1}(\eta(L)) = L$ holds. This is the case if, and only if, L is closed under behavioral equivalence. So, we have to show that L is closed under each equation from Lemma 4.7.2:

Theorem 5.4.8. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet. Then the following problem is decidable:*

Input: *A regular language $L \subseteq \Sigma^*$*
Question: *Is L closed under behavioral equivalence?*

Proof. We prove this with the help of a rational transduction (cf. [Ber79]). To this end, let τ be the following rational transduction:

$$I_\Sigma^* \cup I_\Sigma^* \{(\ell, r), (r, \ell) \mid \ell \equiv r \text{ is an equation in Lemma 4.7.2}\} I_\Sigma^*$$

where $I_\Sigma = \{(\alpha, \alpha) \mid \alpha \in \Sigma\}$ is the identity relation on Σ. Then by Theorem 4.7.8 a language $L \subseteq \Sigma^*$ is closed under $\equiv$ if, and only if, $L = \tau(L)$ holds. If L is a regular language then $\tau(L)$ also is regular. Hence, we can finally check whether $L = \tau(L)$ holds. ◂

Note that for a regular language $L \subseteq \Sigma^*$ with $L = \eta^{-1}(\eta(L))$ the plq language $\eta(L)$ is recognizable. Nevertheless the converse implication is not true in the general case: let

$a, b \in A$ be two distinct letters. Then the plq language $\eta(a^*\overline{b}^*)$ is recognizable but $a^*\overline{b}^*$ is not closed under behavioral equivalence.

5.5 Conclusion

We have considered the rational languages of the partially lossy queue and stack monoid, respectively. Concretely, we have studied several classical decision problems on this class of languages, which are known to be decidable for regular languages.

First, we have generalized a result by Render and Kambites (cf. [RK09]) to partially lossy stacks. In other words, we have seen that a language in the pls monoid is rational if, and only if, the normal forms of the elements in this set build a regular language. Moreover, we can compute this regular language in polynomial time. From this result we infer the closure of the class of rational languages under Boolean operations and the decidability of the rational membership, intersection emptiness, universality, inclusion, and equivalence problem in polynomial time or space. However, until now it is still unknown whether also unique decipherability is decidable.

For the rational plq languages we have shown that the membership problem is decidable using only nondeterministic logarithmic space. We have also seen that all of the other aforementioned problems are undecidable.

Chapter 6
Recognizable Languages

6.1 Introduction

In the previous chapter we have introduced the rational and recognizable languages of a given monoid. Concretely, we have focused on the algorithmic aspects of rational languages in the partially lossy queue and stack monoids. Unfortunately, most of the mentioned decision problems (like intersection emptiness, universality, and recognizability) are undecidable in the plq monoid. However, we can also study the recognizable languages of this monoid (recall that by [McK64] any recognizable language is rational). In this case, all of the aforementioned problems get decidable by known constructions from automata theory. This is also another indication that the rational plq languages are not effectively recognizable. In this chapter we will see another evidence that the class of recognizable languages is a proper subclass of the rational languages in the plq monoid: the class of recognizable plq languages is not closed under product and iteration. We will also see that the classes of recognizable and rational languages in the pls monoid do not coincide as well, since in most cases there are only the two trivial recognizable languages: the monoid itself and the empty set.

Hence, since in all considered cases there are rational languages which are not recognizable, we may ask in which cases a rational language is recognizable. To this end, we will give several characterizations of the recognizable languages. First, we consider those languages in the pls monoid. As mentioned before, for partially lossy stacks with an at least binary lossiness alphabet, there are only two recognizable languages. If the underlying lossiness alphabet is unary, then the recognizable pls languages are recognized by finite, cyclic groups.

Afterwards, we consider the recognizable plq languages. We will first give some kind of rational expressions which generate exactly the recognizable languages. To this end, we need special restrictions to the monoid's product and iteration, which results in the so-called *q-rational* languages. Note that this approach is similar to Ochmański's *c-rational* trace languages [Och85] generating the recognizable trace languages in terms of rational operations (in this case we restrict the iteration to connected trace languages, only).

Additionally, we will present a characterization of the recognizable plq languages in terms of a monadic second-order logic. This logic is derived from another logic introduced by Büchi in [Büc60]. In that logic we understand words as linear orders of positions labeled with the letters of the underlying alphabet. Büchi proved that the monadic second-order logic on this interpretation of words describes exactly the class of regular languages. From this result we obtain an even brighter understanding on how to formalize the behavior of finite automata. A similar logic concerning recognizable trace languages was introduced by Ebinger in [Ebi94, Ebi95]. In this chapter we will give another monadic second-order logic describing the recognizable plq languages.

6.2 Partially Lossy Stacks

First, we consider the recognizable pls languages. In the previous chapter we have learned that a pls language T is rational if, and only if, the set of normal forms of T is a regular language. In other words, we already know a simple characterization of the rational languages. However, one may also ask for a characterization of the recognizable languages. We already know that in the bicyclic monoid $\mathbb{T}(C_1)$ (recall that this is the transformation monoid of a pls $\mathcal{P}_\mathcal{L}$ with a unary lossiness alphabet $\mathcal{L}$), a language is recognizable if, and only if, its syntactic monoid is a finite cyclic group [CP61]. In this section we will give a proof of this result. Additionally, we will show that for any other lossiness alphabet $\mathcal{L} = (F, U)$ (i.e., we have $|F| + |U| \geq 2$) the pls monoid has only trivial recognizable languages.

To this end, we consider the natural homomorphism $\phi\colon \mathbb{T}(\mathcal{P}_\mathcal{L}) \to \mathbb{S}(T)$ into the syntactic monoid of a pls language. This function is even surjective and, hence, an epimorphism. So, let f be any epimorphism into a finite monoid $\mathbb{F}$. We show in the next lemma that the images of the write and read actions of a letter $a \in A$ induce a subgroup in $\mathbb{F}$.

Lemma 6.2.1. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $\mathbb{F}$ be a finite monoid, $f\colon \mathbb{T}(\mathcal{P}_\mathcal{L}) \twoheadrightarrow \mathbb{F}$ be an epimorphism into $\mathbb{F}$, and $a \in A$. Then $f(\{[\![a]\!], [\![\overline{a}]\!]\}^*)$ is a subgroup of $\mathbb{F}$ which is generated by $f([\![a]\!])$. In particular, this subgroup is cyclic and has the identity $f([\![\varepsilon]\!])$.*

Proof. Since $\mathbb{F}$ is a finite monoid, there are numbers $0 \leq m < n$ with $f([\![a^m]\!]) = f([\![a^n]\!])$. Then we observe the following:

$$\begin{aligned}
f([\![\overline{a}]\!]) &= f([\![a^m\overline{a}^m\,\overline{a}]\!]) && \text{(since } a^m\overline{a}^m \equiv \varepsilon \text{ by Lemma 4.4.1(1))}\\
&= f([\![a^n\overline{a}^m\,\overline{a}]\!]) && \text{(since } f([\![a^m]\!]) = f([\![a^n]\!])\text{)}\\
&= f([\![a^{n-m}\,\overline{a}]\!]) && \text{(since } m < n \text{ and } a^m\overline{a}^m \equiv \varepsilon\text{)}\\
&= f([\![a^{n-m-1}]\!])\,. && \text{(since } m < n\text{)}
\end{aligned}$$

Hence, $f(\{[\![a]\!], [\![\overline{a}]\!]\}^*)$ is generated by $f([\![a]\!])$. Moreover, from $a\overline{a} \equiv \varepsilon$ we know that $f([\![\overline{a}]\!])$ is the right-inverse of $f([\![a]\!])$. We finally have to prove that $f([\![\overline{a}]\!])$ also is the left-inverse of $f([\![a]\!])$. This can be shown with the following calculations:

$$\begin{aligned}
f([\![\overline{a}]\!]) \cdot f([\![a]\!]) &= f([\![a^{n-m-1}]\!]) \cdot f([\![a]\!])\\
&= f([\![a^{n-m}]\!])\\
&= f([\![a]\!]) \cdot f([\![a^{n-m-1}]\!])\\
&= f([\![a]\!]) \cdot f([\![\overline{a}]\!]) = f([\![a\overline{a}]\!])\\
&= f([\![\varepsilon]\!])\,.
\end{aligned}$$

Hence, $f([\![a]\!])$ has the inverse element $f([\![\overline{a}]\!])$. Since $f([\![a]\!])$ generates $f(\{[\![a]\!], [\![\overline{a}]\!]\}^*)$, this set is a cyclic group. ◂

From this statement we can finally infer the following characterization of the recognizable counter languages, i.e., of the recognizable pls languages with underlying unary lossiness alphabet $\mathcal{L}$:

Theorem 6.2.2. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F|+|U| = 1$ and $T \subseteq \mathbb{T}(\mathcal{P}_\mathcal{L})$. Then the following statements are equivalent:*

(1) T is recognizable in $\mathbb{T}(\mathcal{P}_\mathcal{L})$.

(2) The syntactic monoid $\mathbb{S}(T)$ of T is a finite cyclic group [CP61].

(3) T is a finite union of pls languages $S_{k,\ell} := \{[\![t]\!] \in \mathbb{T}(\mathcal{P}_\mathcal{L}) \mid |t|_A - |t|_{\overline{A}} \bmod k = \ell\}$ where $k, \ell \in \mathbb{N}$ with $k > \ell$.

Proof. Let $A = F \cup U = \{a\}$. We prove these equivalences with the help of three implications. First, we prove "(3)⇒(1)". To this end, we show that $S_{k,\ell}$ is recognizable for $k, \ell \in \mathbb{N}$ with $k > \ell$. There exists an epimorphism $f\colon \mathbb{T}(\mathcal{P}_\mathcal{L}) \twoheadrightarrow \mathbb{Z}_k$ satisfying $f([\![a]\!]) = 1$ and $f([\![\overline{a}]\!]) = k-1$ (in other words, each basic transformation $[\![a]\!]$ increases a number modulo k and each $[\![\overline{a}]\!]$ decreases this number). Then we can see $f^{-1}(\ell) = S_{k,\ell}$. Hence, $S_{k,\ell}$ is recognizable and, therefore, the finite union T of such languages also is recognizable.

Next, let T be recognizable in $\mathbb{T}(\mathcal{P}_\mathcal{L})$. Then the syntactic monoid $\mathbb{S}(T)$ of T is finite and T is recognized via the natural epimorhism $\phi\colon \mathbb{T}(\mathcal{P}_\mathcal{L}) \twoheadrightarrow \mathbb{S}(T)\colon t \mapsto [t]_\sim$ (where $\sim$ is the syntactic congruence on $\mathbb{T}(\mathcal{P}_\mathcal{L})$). Then due to Lemma 6.2.1 $\phi(\mathbb{T}(\mathcal{P}_\mathcal{L})) = \phi(\{[\![a]\!], [\![\overline{a}]\!]\}^*)$ is a cyclic group.

Finally, assume that $\mathbb{S}(T)$ is a finite cyclic group. In other words, we have $\mathbb{S}(T) \cong \mathbb{Z}_k$ for a number $k \in \mathbb{N} \setminus \{0\}$. Then there is an epimorphism $f\colon \mathbb{T}(\mathcal{P}_\mathcal{L}) \twoheadrightarrow \mathbb{Z}_k$. We can assume that $f([\![a]\!]) = 1$ holds (where $A = \{a\}$) since $\mathbb{Z}_k$ is generated by the element $f([\![a]\!])$ according to Lemma 6.2.1. Since T is recognized by $\mathbb{Z}_k$ via f there is a finite set $X \subseteq \mathbb{Z}_k$ with $f^{-1}(X) = T$. But then we have

$$T = f^{-1}(X) = \bigcup_{\ell \in X} f^{-1}(\ell) = \bigcup_{\ell \in X} S_{k,\ell}\,.$$

◂

Next, we consider the non-unary case. In other words, we consider the recognizable partially lossy stack languages with an underlying at least binary lossiness alphabet $\mathcal{L}$. To this end, we first show that the image of an epimorphism $f\colon \mathbb{T}(\mathcal{P}_\mathcal{L}) \twoheadrightarrow \mathbb{F}$ into a finite monoid $\mathbb{F}$ is the trivial subgroup:

Lemma 6.2.3. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$, $\mathbb{F}$ be a finite monoid with identity e, $f\colon \mathbb{T}(\mathcal{P}_\mathcal{L}) \twoheadrightarrow \mathbb{F}$ be an epimorphism into $\mathbb{F}$, and $a \in A$. Then we have $f([\![a]\!]) = e$.*

Proof. Since $|A| = |F| + |U| \geq 2$ there is a letter $b \in A \setminus \{a\}$. By Lemma 6.2.1 there is a positive number $n \in \mathbb{N} \setminus \{0\}$ with $f([\![b^n]\!]) = f([\![\varepsilon]\!])$. Since $f([\![\overline{b}]\!])$ is the inverse element of $f([\![b]\!])$ we also have $f([\![\overline{b}^n]\!]) = f([\![\varepsilon]\!])$.

Now, we have to distinguish two cases:

(1) $a \in F$ is a forgettable letter. Then we obtain the following:

$$\begin{aligned} f([\![a]\!]) &= f([\![a]\!]) \cdot f([\![\overline{b}^n]\!]) && \text{(since } f([\![\overline{b}^n]\!]) = f([\![\varepsilon]\!])\text{)} \\ &= f([\![a\overline{b}^n]\!]) \\ &= f([\![\overline{b}^n]\!]) && \text{(since } a\overline{b} \equiv \overline{b} \text{ by Lemma 4.4.1(3))} \\ &= f([\![\varepsilon]\!]) = e\,. && \text{(by the choice of } n\text{)} \end{aligned}$$

(2) $a \in U$ is an unforgettable letter. Then we learn

$$\begin{aligned} f([\![a]\!]) &= f([\![a]\!]) \cdot f([\![\overline{b}^n]\!]) && \text{(since } f([\![\overline{b}^n]\!]) = f([\![\varepsilon]\!])\text{)} \\ &= f([\![a\overline{b}^n]\!]) \\ &= f([\![\perp\!\!\!\perp]\!])\,. && \text{(since } a\overline{b} \equiv \perp\!\!\!\perp \text{ by Lemma 4.4.1(2))} \end{aligned}$$

Since $\mathbb{F}$ is finite there is a number $m \in \mathbb{N} \setminus \{0\}$ with $f([\![a^m]\!]) = f([\![\varepsilon]\!])$. Then we finally see

$$f([\![a]\!]) = f([\![\perp\!\!\!\perp]\!]) = f([\![\perp\!\!\!\perp^m]\!]) = f([\![a^m]\!]) = f([\![\varepsilon]\!]) = e\,.$$ ◂

This finally shows that the partially lossy stack monoids with an at least binary lossiness alphabet have only trivial recognizable languages:

Theorem 6.2.4. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ and let $T \subseteq \mathbb{T}(\mathcal{P}_\mathcal{L})$ be a partially lossy stack language. Then T is recognizable if, and only if, the syntactic monoid $\mathbb{S}(T)$ of T is a singleton. In particular, T is recognizable iff $T = \varnothing$ or $T = \mathbb{T}(\mathcal{P}_\mathcal{L})$ holds.*

Proof. Let $T \subseteq \mathbb{T}(\mathcal{P}_\mathcal{L})$ be recognizable. Then its syntactic monoid $\mathbb{S}(T)$ is finite and recognizes T via the natural epimorphism $\phi\colon \mathbb{T}(\mathcal{P}_\mathcal{L}) \twoheadrightarrow \mathbb{S}(T)$. By Lemma 6.2.3 we know $f([\![t]\!]) = e = [\varepsilon]_\sim$ for each action sequence $t \in \Sigma^*$. Hence, by surjectivity of f we infer $\mathbb{S}(T) = \{[\varepsilon]_\sim\}$. Conversely, let $T \subseteq \mathbb{T}(\mathcal{P}_\mathcal{L})$ have a syntactic monoid with $|\mathbb{S}(T)| = 1$. Then this monoid is finite and, hence, T is recognizable. ◂

6.3 Intermezzo: Büchi's Theorem

Next, we want to characterize the recognizable partially lossy queue languages. One of the characterizations is a special logic on transformations of a partially lossy queue. This logic will be in some sense a generalization of Büchi's word logic. Hence, we should first recall this special logic:

Let A be an alphabet. The signature λ[xii] consists of the binary symbol $<$ and the unary symbol Λ_a for each $a \in A$.

Let $w = a_1 \dots a_n$ be a word with $a_1, \dots, a_n \in A$. The *word model* of w is the relational λ-structure

$$\underline{w} := (\mathrm{dom}(w), <^w, (\Lambda_a^w)_{a \in A})$$

[xii] The name λ of this signature stems from the greek phrase "$\lambda\acute{\varepsilon}\xi\eta$" which translates to "word".

where $\mathrm{dom}(w) := \{1, \dots, n\}$ is the set of positions in w, $<^w$ is the natural (strict) ordering on $\mathrm{dom}(w)$, and $\Lambda_a^w := \{i \in \mathrm{dom}(w) \mid a_i = a\}$ is the set of all positions in w labeled with the letter $a \in A$. Note that $(\Lambda_a^w)_{a \in A}$ is a partition of the set $\mathrm{dom}(w)$.

Example 6.3.1. Let $w = abbacb$. Then we have $\mathrm{dom}(w) = \{1, \dots, 6\}$, $\Lambda_a^w = \{1, 4\}$, $\Lambda_b^w = \{2, 3, 6\}$, and $\Lambda_c^w = \{5\}$. ⌟

Now, we are considering formulas over the signature λ. To simplify notation we also write $\Lambda_B(x)$ instead of the formula $\bigvee_{a \in B} \Lambda_a(x)$ for any non-empty $B \subseteq A$. Moreover, we write $x \leq y$ instead of $x < y \vee x = y$.

Let $\phi \in \mathsf{MSO}[\lambda]$ be a sentence (i.e., ϕ contains no free variables). Then the language defined by ϕ is $L(\phi) := \{w \in A^* \mid \underline{w} \vDash \phi\}$. A language $L \subseteq A^*$ is $\mathsf{MSO}[\lambda]$-*definable* if there is a sentence $\phi \in \mathsf{MSO}[\lambda]$ with $L = L(\phi)$. Similarly, L is $\mathsf{FO}[\lambda]$-*definable* if there is a sentence $\phi \in \mathsf{FO}[\lambda]$ with $L = L(\phi)$.

Theorem 6.3.2. *Let A be an alphabet and $L \subseteq A^*$ be a language. Then the following statements hold:*

(1) (Büchi's Theorem) L is regular if, and only if, L is $\mathsf{MSO}[\lambda]$-definable [Büc60].

(2) L is aperiodic if, and only if, L is $\mathsf{FO}[\lambda]$-definable [MP71]. ◂

6.4 Partially Lossy Queues

We have already seen that it is undecidable whether a given rational language is recognizable (cf. Theorem 5.4.7(2)). However, we want to characterize in which cases a rational language is recognizable. Concretely, we give some rational-like expressions which are fully describing the recognizable plq languages. To this end, we have to introduce some special restrictions to the concatenation and iteration of recognizable languages. Additionally, we have to add a non-monotonic operation to our expressions: the complement operation. We call languages constructed via these restricted operations the *q-rational* languages in $\mathbb{T}(Q_{\mathcal{A}})$ (the definition of this notion can be found at Page 104). With the help of this definition we will obtain a characterization in the manner of Kleene [Kle51]. Additionally, we want to translate Büchi's Theorem [Büc60] to the plq monoid. We do this with the help of special modifications of Büchi's word logic $\mathsf{MSO}[\lambda]$ which we have defined in the previous section. Concretely, our new logic $\mathsf{MSO}[o]$ still identifies an action sequence $w \in \Sigma^*$ as a set of positions labeled with letters from Σ. But in contrast to the word logic we have to restrict comparisons between write and read actions to ensure that our logic only describes recognizable plq languages. The full definition of this logic can be found at Page 107.

Now, we state the main theorem of this section. As mentioned before, we will give the concrete definitions of q-rational languages and the logic $\mathsf{MSO}[o]$ later in this section. We prove this theorem with the help of three implications in Propositions 6.4.22, 6.4.28 and 6.4.31.

Theorem 6.4.1. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F|+|U| \geq 2$ and $T \subseteq \mathbb{T}(Q_{\mathcal{L}})$. Then the following statements are effectively equivalent:*

(A) T is recognizable.

(B) T is q-rational.

(C) T is $\mathsf{MSO}[o]$-definable.

6.4.1 Some Helpful Characterizations

Before diving into the proof of Theorem 6.4.1 we prove two further characterizations of the recognizable plq languages which turned out to be convenient for the simplification of our proof. We already know these characterizations from Huschenbett et al. [HKZ17] which were given there for the transformation monoid $\mathbb{T}(Q_{(\varnothing,A)})$ of a reliable queue. Here, we generalize these equivalences to plq monoids $\mathbb{T}(Q_{\mathcal{L}})$ with arbitrary lossiness alphabet $\mathcal{L} = (F, U)$.

Concretely, we prove the correspondence of recognizability in the plq monoid to regularity in the underlying free monoid. We also describe the recognizable languages in terms of Boolean combinations of plq languages $\mathrm{wrt}^{-1}(R)$, $\mathrm{rd}^{-1}(R)$, and some special languages Ω_ℓ for regular languages $R \subseteq A^*$ and numbers $\ell \in \mathbb{N}$:

Definition 6.4.2. Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $t \in \mathbb{T}(Q_{\mathcal{L}})$. Then the *overlap's bounded width* of t is

$$\mathrm{obw}(t) := \inf\left\{|\mathrm{rd}_2(s)| \;\middle|\; s \in \mathbb{T}(Q_{\mathcal{L}}), \mathrm{wrt}(s) = \mathrm{wrt}(t), \mathrm{rd}(s) = \mathrm{rd}(t), |\mathrm{rd}_2(s)| > |\mathrm{rd}_2(t)|\right\}.$$

Similarly, for $w \in \Sigma^*$ we may define $\mathrm{obw}(w) := \mathrm{obw}([\![w]\!])$. Furthermore, for $\ell \in \mathbb{N}$ set

$$\Omega_\ell := \{t \in \mathbb{T}(Q_{\mathcal{L}}) \mid \mathrm{obw}(t) > \ell\}.$$

The overlap's bounded width specifies the minimal length of the overlap of a transformation having the same projections and a longer overlap. If no such word exists then we set this value to ∞. In particular, we have $t \in \Omega_\ell$ for a transformation $t \in \mathbb{T}(Q_{\mathcal{L}})$ and a number $\ell \in \mathbb{N}$ if, and only if, any transformation $s \in \mathbb{T}(Q_{\mathcal{L}})$ having the same projections (i.e., $\mathrm{wrt}(s) = \mathrm{wrt}(t)$ and $\mathrm{rd}(s) = \mathrm{rd}(t)$) and a longer overlap (i.e., $|\mathrm{rd}_2(s)| > |\mathrm{rd}_2(t)|$) also satisfies $|\mathrm{rd}_2(s)| > \ell$.

Example 6.4.3. Let $\mathcal{L} = (\varnothing, \{a, b\})$ and $t = [\![\overline{abab}a\bar{a}b\bar{b}abab]\!]$. Then there are two transformations with the same projections and longer overlaps:

$$s_1 = [\![\overline{ab}a\bar{a}b\bar{b}a\bar{a}b\bar{b}ab]\!] \qquad \text{and} \qquad s_2 = [\![a\bar{a}b\bar{b}a\bar{a}b\bar{b}a\bar{a}b\bar{b}]\!].$$

We have $|\mathrm{rd}_2(s_1)| = 4$ and $|\mathrm{rd}_2(s_2)| = 6$. Therefore we obtain $\mathrm{obw}(t) = 4$, $\mathrm{obw}(s_1) = 6$, and $\mathrm{obw}(s_2) = \infty$. Hence, $t \in \Omega_3 \setminus \Omega_4$ holds.

From [HKZ17, Observation 9.1] we already know that the reliable queue languages $\{t \in \mathbb{T}(Q_{(\varnothing,U)}) \mid |\mathrm{rd}_2(t)| > \ell\}$ are not recognizable for any number $\ell \in \mathbb{N}$. This means, there is no non-trivial property of the overlap's width $|\mathrm{rd}_2(t)|$ which is recognizable in the reliable queue monoid $\mathbb{T}(Q_{(\varnothing,U)})$. An appropriate alternative for the generators of the Boolean

algebra of recognizable plq languages has been found in such kind of "over-approximation" of the overlap's length (note that $\mathrm{obw}(t) > |\mathrm{rd}_2(t)|$ holds). Additionally, the following two observations provide more motivation of this notion. First, we show that the characteristic $\chi(t)$ of the transformation $t \in \mathbb{T}(\mathcal{Q}_\mathcal{L})$ is fully described by $\mathrm{wrt}(t)$, $\mathrm{rd}(t)$, and $\mathrm{obw}(t)$.

Observation 6.4.4. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ and $s, t \in \mathbb{T}(\mathcal{Q}_\mathcal{L})$. Then we have* $\mathrm{wrt}(s) = \mathrm{wrt}(t)$, $\mathrm{rd}(s) = \mathrm{rd}(t)$, *and* $\mathrm{obw}(s) = \mathrm{obw}(t)$ *if, and only if, $s = t$ holds.*

Proof. The implication "$\Leftarrow$" is obvious. So, we only have to prove the converse implication. Let $s, t \in \mathbb{T}(\mathcal{Q}_\mathcal{L})$ be two transformations with $\mathrm{wrt}(s) = \mathrm{wrt}(t)$, $\mathrm{rd}(s) = \mathrm{rd}(t)$, and $\mathrm{obw}(s) = \mathrm{obw}(t)$. Due to Theorem 4.7.8 it suffices to show $\mathrm{rd}_2(s) = \mathrm{rd}_2(t)$.

By definition of $\mathrm{obw}(.)$ we know

$$|\mathrm{rd}_2(s)| < \mathrm{obw}(s) = \mathrm{obw}(t) = \inf \left\{ |\mathrm{rd}_2(u)| \;\middle|\; \begin{array}{c} u \in \mathbb{T}(\mathcal{Q}_\mathcal{L}), \mathrm{wrt}(u) = \mathrm{wrt}(t), \mathrm{rd}(u) = \mathrm{rd}(t), \\ |\mathrm{rd}_2(u)| > |\mathrm{rd}_2(t)| \end{array} \right\}.$$

Recall that $\mathrm{wrt}(s) = \mathrm{wrt}(t)$ and $\mathrm{rd}(s) = \mathrm{rd}(t)$ holds. Then $|\mathrm{rd}_2(s)| > |\mathrm{rd}_2(t)|$ would imply $\mathrm{obw}(t) \leq |\mathrm{rd}_2(s)| < \mathrm{obw}(s) = \mathrm{obw}(t)$, which is impossible. Hence, we have $|\mathrm{rd}_2(s)| \leq |\mathrm{rd}_2(t)|$. By symmetry we also obtain $|\mathrm{rd}_2(s)| \geq |\mathrm{rd}_2(t)|$. In other words, $\mathrm{rd}_2(s)$ and $\mathrm{rd}_2(t)$ are suffixes of $\mathrm{rd}(s) = \mathrm{rd}(t)$ having the same length, i.e., we have $\mathrm{rd}_2(s) = \mathrm{rd}_2(t)$. This implies $\chi(s) = \chi(t)$ and, therefore, $s = t$. ◂

This means, similar to the characteristic $\chi(t)$, the triple $(\mathrm{wrt}(t), \mathrm{rd}(t), \mathrm{obw}(t))$ is another kind of characterization of the transformation $t \in \mathbb{T}(\mathcal{Q}_\mathcal{L})$.

The following observation proves that from a transformation $t \in \mathbb{T}(\mathcal{Q}_\mathcal{L})$ with small overlap's bounded width we obtain another transformation $s \in \mathbb{T}(\mathcal{Q}_\mathcal{L})$ which has an overlap $\mathrm{rd}_2(s)$ of this bounded width. This knowledge is very helpful to prove Theorem 6.4.1 and the succeeding Theorem 6.4.6.

Observation 6.4.5. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$, $\ell \in \mathbb{N}$, and $t \in \mathbb{T}(\mathcal{Q}_\mathcal{L})$. Then* $\mathrm{obw}(t) \leq \ell$ *if, and only if, there is $r_2 \in A^{\leq \ell}$ with* $\mathrm{rd}(t) \in A^* r_2$ *and $w_1 \in \mathrm{redsup}_\mathcal{L}(r_2)$ for a prefix w_1 of* $\mathrm{wrt}(t)$ *such that* $|\mathrm{rd}_2(t)| < |r_2|$.

Proof. We first suppose $\mathrm{obw}(t) > \ell$. Let $r_2 \in A^{\leq \ell}$ such that $\mathrm{rd}(t) \in A^* r_2$ and $w_1 \in \mathrm{redsup}_\mathcal{L}(r_2)$ for a prefix w_1 of $\mathrm{wrt}(t)$. Furthermore, let $r_1 \in A^*$ with $\mathrm{rd}(t) = r_1 r_2$. Set $s := [\![\overline{r_1}\mathrm{wrt}(t)\overline{r_2}]\!]$. Then we have $\mathrm{wrt}(s) = \mathrm{wrt}(t)$, $\mathrm{rd}(s) = \mathrm{rd}(t)$, and $|\mathrm{rd}_2(s)| \leq |r_2| \leq \ell$. Since $w_1 \in \mathrm{redsup}_\mathcal{L}(r_2)$ and w_1 is a prefix of $\mathrm{wrt}(t) = \mathrm{wrt}(s)$, we have $\mathrm{rd}_2(s) = r_2$ according to Corollary 4.7.15. Since $|\mathrm{rd}_2(t)| < |\mathrm{rd}_2(s)|$ would imply $\mathrm{obw}(t) \leq |\mathrm{rd}_2(s)| \leq \ell$, we have $|\mathrm{rd}_2(t)| \geq |\mathrm{rd}_2(s)| = |r_2|$.

Now assume $\mathrm{obw}(t) \leq \ell$. Then there is $s \in \mathbb{T}(\mathcal{Q}_\mathcal{L})$ with $\mathrm{wrt}(s) = \mathrm{wrt}(t)$, $\mathrm{rd}(s) = \mathrm{rd}(t)$, and $|\mathrm{rd}_2(t)| < |\mathrm{rd}_2(s)| \leq \ell$. Consider $r_2 := \mathrm{rd}_2(s)$. Then we have $|r_2| = |\mathrm{rd}_2(s)| \leq \ell$, $\mathrm{rd}(t) = \mathrm{rd}(s) \in A^* r_2$, and $w_1 \in \mathrm{redsup}_\mathcal{L}(r_2)$ for a prefix $w_1 \in A^*$ of $\mathrm{wrt}(s) = \mathrm{wrt}(t)$. ◂

With the help of the overlap's bounded width and the knowledge from the previous observations we are able to state the following equivalences:

Theorem 6.4.6. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F|+|U| \geq 2$ and $T \subseteq \mathbb{T}(Q_{\mathcal{L}})$. Then the following statements are effectively equivalent:*

(i) *T is recognizable.*

(ii) *$\eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ is regular.*

(iii) *T is a Boolean combination of plq languages of the form $\mathrm{wrt}^{-1}(R)$, $\mathrm{rd}^{-1}(R)$, and Ω_ℓ for regular languages $R \subseteq A^*$ and numbers $\ell \in \mathbb{N}$.*

Note that $\eta^{-1}(T) \subseteq \Sigma^*$ is the union of the equivalence classes of the behavioral equivalence in T. From Corollary 4.7.15 we also know that each such equivalence class contains a word from $\overline{A}^* A^* \overline{A}^*$. Hence, the language $\eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ contains at least one representative from each equivalence class of the behavioral equivalence $\equiv$ in T.

Remark 6.4.7. Let $\mathcal{L} = (\varnothing, U)$ be a lossiness alphabet with $|U| \geq 2$ (i.e., we are considering a reliable queue). Huschenbett et al. proved in [HKZ17] that we can extend Theorem 6.4.6 by the following item:

(iv) $\eta^{-1}(T) \cap A^* \overline{A}^* A^*$ is regular.

However, this theorem cannot be expanded by the statement "$\eta^{-1}(T) \cap L$ is regular" where L is either $\overline{A}^* A^*$, $A^* \overline{A}^*$, or the union of both languages.

Moreover, this fourth item is not equivalent for other lossiness alphabets. So, let $\mathcal{L} = (F, U)$ with $F \neq \varnothing$ and $|F| + |U| \geq 2$. For two distinct letters $a, b \in A$ with $a \in F$ the language $T := \{[\![\overline{a}^n a^n b \overline{b}]\!] \mid n \in \mathbb{N}\}$ is not recognizable since

$$\eta^{-1}(T) = \{\overline{a}^n a^n b \overline{b} \mid n \in \mathbb{N}\}$$

is no regular language. However, the language $\eta^{-1}(T) \cap A^* \overline{A}^* A^* = \{b\overline{b}\}$ is finite and, therefore, regular. ⌟

The Implication "(i)⇒(ii)" in Theorem 6.4.6

The first implication in Theorem 6.4.6 that we want to prove is very simple:

Proposition 6.4.8. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $T \subseteq \mathbb{T}(Q_{\mathcal{L}})$ be recognizable. Then $\eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ is regular.*

Proof. Since T is recognizable, the language $\eta^{-1}(T)$ is recognizable and hence regular. Since the class of regular languages is closed under intersection, $\eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ is regular. ◂

The Implication "(ii)⇒(iii)" in Theorem 6.4.6

Towards the proof of the second implication in Theorem 6.4.6 we fix an arbitrary lossiness alphabet $\mathcal{L} = (F, U)$ with $|F| + |U| \geq 2$. Let $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ be a plq language such that $\eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ is regular. Now, we will partition T into two plq languages: $T \cap \Omega_\ell$ and $T \cap \mathbb{T}(\mathcal{Q}_\mathcal{L}) \setminus \Omega_\ell$ for an appropriate natural number $\ell \in \mathbb{N}$. We will show then that both plq languages satisfy property (iii) of Theorem 6.4.6. This means, both languages are a Boolean combination of plq languages of the form $\mathrm{wrt}^{-1}(R)$, $\mathrm{rd}^{-1}(R)$, and Ω_k for regular languages $R \subseteq A^*$ and numbers $k \in \mathbb{N}$. To prove this result, we first show that any transformation $t \in \Omega_k$ with $k \in \mathbb{N}$ having at least k read actions has the same behavior as another one ending with k read actions.

Lemma 6.4.9. *Let $t \in \mathbb{T}(\mathcal{Q}_\mathcal{L})$ and $r_1, r_2 \in A^*$ with $\mathrm{rd}(t) = r_1 r_2$ and $|\mathrm{rd}_2(t)| \leq |r_2| < \mathrm{obw}(t)$. Then we have $t = [\![\overline{r_1}\mathrm{wrt}(t)\overline{r_2}]\!]$.*

Proof. If $\mathrm{obw}(t) < \infty$ holds, this number is the length of the shortest suffix $s \in A^*$ of $\mathrm{rd}(t)$ which is longer than $\mathrm{rd}_2(t)$ and satisfies $p \in \mathrm{redsup}_\mathcal{L}(s)$ for a prefix p of $\mathrm{wrt}(t)$. By minimality of $\mathrm{obw}(t)$ we obtain $\mathrm{sws}(\mathrm{wrt}(t), r_2) = \mathrm{sws}(\mathrm{wrt}(t), \mathrm{rd}_2(t)) = \mathrm{rd}_2(t)$ for each suffix r_2 of $\mathrm{rd}(t)$ with $|\mathrm{rd}_2(t)| \leq |r_2| < \mathrm{obw}(t)$. Hence, application of Lemma 4.7.14 and Corollary 4.7.15 yields

$$t = [\![\overline{\mathrm{rd}_1(t)}\mathrm{wrt}(t)\overline{\mathrm{rd}_2(t)}]\!] = [\![\overline{r_1}\mathrm{wrt}(t)\overline{r_2}]\!]$$

where r_1 is the complementary prefix of $\mathrm{rd}(t)$ wrt. r_2.

If otherwise $\mathrm{obw}(t) = \infty$, we know $w_1 \notin \mathrm{redsup}_\mathcal{L}(r_2)$ for any prefix $w_1 \in A^*$ of $\mathrm{wrt}(t)$ and for any suffix $r_2 \in A^*$ of $\mathrm{rd}(t)$ which is longer than $\mathrm{rd}_2(t)$. This implies $\mathrm{sws}(\mathrm{wrt}(t), r_2) = \mathrm{rd}_2(t)$ in this case. Hence, utilization of Lemma 4.7.14 and Corollary 4.7.15 results in

$$t = [\![\overline{\mathrm{rd}_1(t)}\mathrm{wrt}(t)\overline{\mathrm{rd}_2(t)}]\!] = [\![\overline{r_1}\mathrm{wrt}(t)\overline{r_2}]\!]$$

where r_1 is the complementary prefix of $\mathrm{rd}(t)$ wrt. r_2. ◂

Now, we prove that the aforementioned partitions of $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ coincide with Boolean combinations of languages $\mathrm{wrt}^{-1}(R)$, $\mathrm{rd}^{-1}(R)$, and Ω_k for regular languages $R \subseteq A^*$ and numbers $k \in \mathbb{N}$. Note that by swapping projections this lemma generalizes the proofs of [HKZ17, Lemmas 9.9-9.11] which state these results for the reliable queue monoid, only.

Lemma 6.4.10. *Let $\ell \in \mathbb{N}$ and $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ such that $\eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ is recognized by a monoid with ℓ elements. Then the following two plq languages satisfy property (iii) of Theorem 6.4.6:*

(1) $T \cap \Omega_\ell$ and

(2) $T \cap \mathbb{T}(\mathcal{Q}_\mathcal{L}) \setminus \Omega_\ell$.

Proof. Let $L = \eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ be recognized by the finite monoid $\mathbb{F}$ with $\ell := |\mathbb{F}|$ via the homomorphism $\phi\colon \Sigma^* \to \mathbb{F}$. Furthermore, define $\mu, \overline{\mu}\colon A^* \to \mathbb{F}$ such that $\mu(w) = \phi(w)$ and $\overline{\mu}(w) = \phi(\overline{w})$.

(1) ▷ *Idea.* We first consider those transformations $t \in \mathbb{T}(Q_{\mathcal{L}})$ satisfying $\mathrm{obw}(t) > \ell$. If t contains only few read actions (i.e., $|\mathrm{rd}(t)| < \ell$), then Lemma 6.4.9 yields $t = [\![\mathrm{wrt}(t)\overline{\mathrm{rd}(t)}]\!]$. Hence, we have $t \in T$ if, and only if, $\mathrm{wrt}(t)\overline{\mathrm{rd}(t)} \in L$ holds. Otherwise we have $|\mathrm{rd}(t)| \geq \ell$. In this case Lemma 6.4.9 yields the existence of words $x, y, z, r_2 \in A^*$ with $|r_2| = \ell$ and $t = [\![\overline{x}y\overline{zr_2}]\!]$. We will show in Claim 1 that in this case we have

$$t \in T \iff \overline{x}y\overline{zr_2} \in L \iff \overline{xz}y\overline{r_2} \in L\,.$$

In both cases the latter property can be expressed as described in property (iii) of Theorem 6.4.6. ◁

We will show the first statement by establishing the following equation:

$$T \cap \Omega_\ell = \bigcup_{\substack{r_2 \in A^{<\ell},\beta\in\mathbb{F}:\\ \beta\overline{\mu}(r_2)\in\phi(L)}} \mathrm{rd}^{-1}(r_2) \cap \mathrm{wrt}^{-1}\big(\mu^{-1}(\beta)\big) \cap \Omega_\ell \;\cup \bigcup_{\substack{r_2 \in A^{\ell},\alpha,\beta\in\mathbb{F}:\\ \alpha\beta\overline{\mu}(r_2)\in\phi(L)}} \mathrm{rd}^{-1}\big(\overline{\mu}^{-1}(\alpha)r_2\big) \cap \mathrm{wrt}^{-1}\big(\mu^{-1}(\beta)\big) \cap \Omega_\ell\,.$$

We denote the left- and right-hand side of this equation by Y and Z, respectively. Clearly, we have $Y, Z \subseteq \Omega_\ell$. Hence, it suffices to show, given $t \in \Omega_\ell$, that $t \in Y$ holds if, and only if, $t \in Z$ holds. Towards this equivalence we first have to prove the following claim:

▷ *Claim 1.* *Let $t \in \Omega_\ell$, $r_2 \in A^*$ be the longest suffix of $\mathrm{rd}(t)$ with $|r_2| \leq \ell$, and $r_1 \in A^*$ be the complementary prefix of $\mathrm{rd}(t)$ wrt. r_2. Then we have $t \in T$ if, and only if, $[\![\overline{r_1}\mathrm{wrt}(t)\overline{r_2}]\!] \in T$.*

Proof. If $|\mathrm{rd}(t)| < \ell$ holds, we have $r_2 = \mathrm{rd}(t)$ and $r_1 = \varepsilon$ by the choice of r_1 and r_2. Since $\mathrm{obw}(t) > \ell > |\mathrm{rd}(t)|$ holds, we have $\mathrm{obw}(t) = \infty$ in this case. Then from Lemma 6.4.9 we learn $t = [\![\mathrm{wrt}(t)\overline{\mathrm{rd}(t)}]\!] = [\![\overline{r_1}\mathrm{wrt}(t)\overline{r_2}]\!]$ and we are done. So, from now on we assume $|\mathrm{rd}(t)| \geq \ell$. In this case, we have $r_2 \in A^\ell$. Then there are words $x, y, z \in A^*$ with $t = [\![\overline{x}y\overline{zr_2}]\!]$ from Corollary 4.7.15 (if $|\mathrm{rd}_2(t)| > |r_2|$) resp. $\mathrm{obw}(t) > \ell$ and Lemma 6.4.9 (if $|\mathrm{rd}_2(t)| \leq |r_2|$). Note that $r_1 = xz$ and $\mathrm{wrt}(t) = y$ holds. Moreover, due to $|\mathbb{F}| = \ell$ there is $y_0 \in A^{\leq\ell}$ with $\phi(y_0) = \phi(y)$. Then from $|y_0| \leq \ell = |r_2|$ we obtain $\overline{x}y_0\overline{zr_2} \equiv \overline{xz}y_0\overline{r_2}$ according to Lemma 4.7.14. We see:

$$\begin{aligned}
t \in T &\iff \phi(\overline{x}y\overline{zr_2}) \in \phi(L) && \text{(since } t = [\![\overline{x}y\overline{zr_2}]\!]\text{)}\\
&\iff \phi(\overline{x}y_0\overline{zr_2}) \in \phi(L) && \text{(since } \phi(y) = \phi(y_0)\text{)}\\
&\iff [\![\overline{x}y_0\overline{zr_2}]\!] \in T\\
&\iff [\![\overline{xz}y_0\overline{r_2}]\!] \in T && \text{(since } \overline{x}y_0\overline{zr_2} \equiv \overline{xz}y_0\overline{r_2}\text{)}\\
&\iff \phi(\overline{xz}y_0\overline{r_2}) \in \phi(L)\\
&\iff \phi(\overline{xz}y\overline{r_2}) \in \phi(L) && \text{(since } \phi(y) = \phi(y_0)\text{)}\\
&\iff [\![\overline{r_1}\,\mathrm{wrt}(t)\,\overline{r_2}]\!] \in T\,.
\end{aligned}$$

◁

Now, let $t \in \Omega_\ell$, $r_2 \in A^*$ be the longest suffix of $\mathrm{rd}(t)$ with $|r_2| \leq \ell$, and $r_1 \in A^*$ be the complementary prefix of $\mathrm{rd}(t)$ wrt. r_2. Then we have:

$$\begin{aligned}
t \in Y &\iff t \in T\\
&\iff [\![\overline{r_1}\,\mathrm{wrt}(t)\,\overline{r_2}]\!] \in T && \text{(by Claim 1)}\\
&\iff \phi(\overline{r_1}\,\mathrm{wrt}(t)\,\overline{r_2}) \in \phi(L)\\
&\iff \overline{\mu}(r_1)\,\mu(\mathrm{wrt}(t))\,\overline{\mu}(r_2) \in \phi(L)\\
&\iff t \in Z\,.
\end{aligned}$$

(2) ▷ *Idea.* Now, we consider transformations $t \in \mathbb{T}(\mathcal{Q}_\mathcal{L})$ with $\mathrm{obw}(t) \leq \ell$. Let $k := \mathrm{obw}(t) - 1 < \ell$. Then Lemma 6.4.9 yields the existence of words $x, y, r_2 \in A^*$ with $|r_2| = k$ and $t = [\![\overline{x}\, y \overline{r_2}]\!]$. Then we obtain $t \in T$ if, and only if, $\overline{x}\, y \overline{r_2} \in L$. Again, the latter property is expressible as described in property (iii) of Theorem 6.4.6. ⌟

Concretely, we show the second statement by establishing the following equation:

$$T \cap \mathbb{T}(\mathcal{Q}_\mathcal{L}) \setminus \Omega_\ell = \bigcup_{0 \leq k < \ell} \bigcup_{\substack{r_2 \in A^k, \alpha, \beta \in \mathbb{F}: \\ \alpha\beta\overline{\mu}(r_2) \in \phi(L)}} \mathrm{rd}^{-1}\big(\overline{\mu}^{-1}(\alpha) r_2\big) \cap \mathrm{wrt}^{-1}\big(\mu^{-1}(\beta)\big) \cap \Omega_k \setminus \Omega_{k+1}\,.$$

We denote the left- and right-hand side of this equation by Y and Z. From $\mathbb{T}(\mathcal{Q}_\mathcal{L}) = \Omega_0 \supseteq \Omega_1 \supseteq \Omega_2 \supseteq \ldots$ we obtain that $\Omega_k \setminus \Omega_{k+1} \subseteq \Omega_0 \setminus \Omega_\ell = \mathbb{T}(\mathcal{Q}_\mathcal{L}) \setminus \Omega_\ell$ holds for each $0 \leq k < \ell$. Hence, we have $Y, Z \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L}) \setminus \Omega_\ell$. Then it suffices to show, given $t \in \mathbb{T}(\mathcal{Q}_\mathcal{L}) \setminus \Omega_\ell$ that $t \in Y$ holds if, and only if, $t \in Z$ holds.

From $t \notin \Omega_\ell$ we obtain $\mathrm{obw}(t) \leq \ell$. Then there is $0 \leq k < \ell$ with $k + 1 = \mathrm{obw}(t)$ (i.e., we have $t \in \Omega_k \setminus \Omega_{k+1}$). By Lemma 6.4.9 we obtain $t = [\![\overline{r_1} w \overline{r_2}]\!]$ where $\mathrm{wrt}(t) = w$, $\mathrm{rd}(t) = r_1 r_2$, and $|r_2| = k$. Then we learn:

$$t \in Y \iff \phi(\overline{r_1} w \overline{r_2}) \in \phi(L) \iff \overline{\mu}(r_1)\, \mu(w)\, \overline{\mu}(r_2) \in \phi(L) \iff t \in Z\,. \qquad \blacktriangleleft$$

Finally, we infer the implication "(ii)⇒(iii)" in Theorem 6.4.6:

Proposition 6.4.11. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ and $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ such that $\eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ is regular. Then T is a Boolean combination of plq languages of the form $\mathrm{wrt}^{-1}(R)$, $\mathrm{rd}^{-1}(R)$, and Ω_ℓ for regular languages $R \subseteq A^*$ and numbers $\ell \in \mathbb{N}$.*

Proof. Let $\eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ be recognized by a finite monoid with ℓ elements. By set theory we have $T = (T \cap \Omega_\ell) \cup (T \cap \mathbb{T}(\mathcal{Q}_\mathcal{L}) \setminus \Omega_\ell)$. Due to Lemma 6.4.10 the right-hand side of this equation is a finite union of plq languages satisfying property (iii) of Theorem 6.4.6. ◀

The Implication "(iii)⇒(i)" in Theorem 6.4.6

Again, fix a lossiness alphabet $\mathcal{L} = (F, U)$ with $|F| + |U| \geq 2$. We have to prove that $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ is recognizable if it is a Boolean combination of plq languages of the form $\mathrm{wrt}^{-1}(R)$, $\mathrm{rd}^{-1}(R)$, and Ω_ℓ for regular languages $R \subseteq A^*$ and numbers $\ell \in \mathbb{N}$. Since the projections wrt and rd are homomorphisms, we obtain recognizability of $\mathrm{wrt}^{-1}(R)$ and $\mathrm{rd}^{-1}(R)$ if $R \subseteq A^*$ is regular. Hence, we only have to show that Ω_ℓ is recognizable for any $\ell \in \mathbb{N}$. To this end, we prove that $\eta^{-1}(\Omega_\ell)$ is $\mathsf{FO}[\lambda]$-definable and therefore - due to McNaughton and Papert's Theorem [MP71] - even aperiodic.

We first define two $\mathsf{FO}[\lambda]$-formulas embed_ℓ and $\mathsf{overlap}_\ell$ for natural numbers $\ell \in \mathbb{N}$ which describe the following properties: the word model $\underline{w}$ of a queue action sequence $w \in \Sigma^*$ satisfies embed_ℓ if, and only if, there is a suffix r_2 of $\mathrm{rd}(w)$ of length ℓ and a prefix w_1 of $\mathrm{wrt}(w)$ such that r_2 is an $\mathcal{L}$-subword of w_1. In other words, there exists a word $v \in \Sigma^*$ with the same projections and an overlap of length ℓ.

The formula $\mathsf{overlap}_\ell$ strengthens this as follows: $\underline{w}$ satisfies $\mathsf{overlap}_\ell$ if, and only if, there is such word $v \in \Sigma^*$ with the same projections and overlap of length ℓ (i.e., $\underline{w}$ satisfies embed_ℓ) and the overlap's length of w is at least ℓ.

We do this by assigning the last ℓ read actions of w to variables $x_1, \dots, x_\ell$ (where x_1 is the position of the right-most read action in w and x_ℓ is the ℓ^{th} last read action) and the corresponding write actions to variables $y_1, \dots, y_\ell$.

So, let $\ell \in \mathbb{N}$ and $x_1, \dots, x_\ell, y_1, \dots, y_\ell$ be variables. Then we define the following formulas:

(1) $\phi_1 := x_\ell < x_{\ell-1} < \cdots < x_1 \wedge y_\ell < y_{\ell-1} < \cdots < y_1$ - This formula guarantees that the x_i's are mutually distinct and in descending order and the same holds for the y_i's.

(2) $\phi_2 := \bigwedge_{i=1}^{\ell} \bigvee_{a \in A} (\Lambda_{\overline{a}}(x_i) \wedge \Lambda_a(y_i))$ - This formula ensures that x_i reads the same letter from the queue as y_i writes into it.

(3) $\phi_3 := \forall z\colon \big((x_\ell \leq z \wedge \Lambda_{\overline{A}}(z)) \to \bigvee_{i=1}^{\ell} x_i = z\big)$ - Satisfaction of this formula requires the x_i's to be the last ℓ read actions in w.

(4) $\phi_4 := \bigwedge_{i=1}^{\ell-1} \bigwedge_{a \in A} \forall z\colon \big((y_{i+1} < z < y_i \wedge \Lambda_a(y_i)) \to \Lambda_{\overline{A \cup (F \setminus \{a\})}}(z)\big)$ and
$\phi_5 := \bigwedge_{a \in A} \forall z\colon \big((z < y_\ell \wedge \Lambda_a(y_\ell)) \to \Lambda_{\overline{A \cup (F \setminus \{a\})}}(z)\big)$ - These formulas assure that the infix $w[y_{i+1} + 1, y_i - 1]$ contains neither the same letter as y_i nor any unforgettable letter. Hence, together with the formulas above, these ones enforce the last ℓ read actions to be an $\mathcal{L}$-subword of a prefix of the write actions.

(5) $\phi_6 := \bigwedge_{i=1}^{\ell} y_i < x_i$ - This formula guarantees that each x_i appears right from y_i.

By conjunction of the formulas from above we obtain the announced formulas:

$$\mathsf{embed}_\ell := \exists x_1, \dots, x_\ell, y_1, \dots, y_\ell\colon \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5\,,$$
$$\mathsf{overlap}_\ell := \exists x_1, \dots, x_\ell, y_1, \dots, y_\ell\colon \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5 \wedge \phi_6\,.$$

Example 6.4.12. Let $w = abba\overline{aba}$. If $U = \{a\}$ then we have $w \in L(\mathsf{overlap}_1)$ and $w \in L(\mathsf{overlap}_3)$, but $w \notin L(\mathsf{overlap}_2)$ since each assignment of y_2 to b violates ϕ_5 because of the leading a in w.

If $U = \{b\}$ then we have $w \in L(\mathsf{overlap}_1)$, $w \notin L(\mathsf{overlap}_2)$, and $w \notin L(\mathsf{overlap}_3)$ since each assignment of y_1 to a violates ϕ_4 because of the b at the third position in w.

Now, consider $w' = \overline{aba}abba$. Then we have $w' \in L(\mathsf{embed}_\ell)$ if, and only if, $w \in L(\mathsf{embed}_\ell)$ for any $\ell \in \mathbb{N}$. However, we have $w' \in L(\mathsf{overlap}_0)$ and $w' \notin L(\mathsf{overlap}_\ell)$ for each $\ell > 0$. ⌟

In the following lemma we describe the words satisfying the formulas embed_ℓ and $\mathsf{overlap}_\ell$ for any $\ell \in \mathbb{N}$. As announced before, in the first case these are the words where the last ℓ read actions are an $\mathcal{L}$-subword of a prefix of their write actions. Furthermore, the words satisfying $\mathsf{overlap}_\ell$ also satisfy embed_ℓ and have an overlap of at least ℓ symbols.

Lemma 6.4.13. *Let $\ell \in \mathbb{N}$ and $w \in \Sigma^*$. Then the following statements hold:*

(1) $w \in L(\mathsf{embed}_\ell)$ if, and only if, there is $r_2 \in A^\ell$ with $\mathrm{rd}(w) \in A^* r_2$ and $w_1 \in \mathrm{redsup}_{\mathcal{L}}(r_2)$ for a prefix w_1 of $\mathrm{wrt}(w)$.

(2) $w \in L(\mathsf{overlap}_\ell)$ if, and only if, there is $r_2 \in A^\ell$ with $\mathrm{rd}_2(w) \in A^* r_2$ and $w_1 \in \mathrm{redsup}_{\mathcal{L}}(r_2)$ for a prefix w_1 of $\mathrm{wrt}(w)$.

Proof.

(1) First, let $w \in L(\mathsf{embed}_\ell)$. Then there are letters $a_1, \dots, a_\ell, b_1, \dots, b_\ell \in \Sigma$ contained in w such that their positions $p_1, \dots, p_\ell, q_1, \dots, q_\ell \in \mathrm{dom}(w)$ satisfy ϕ_1-ϕ_5. By ϕ_2 we have $a_\ell \dots a_1 = \overline{b_\ell \dots b_1}$. Due to ϕ_1 and ϕ_3 we have $\mathrm{rd}(w) \in A^* b_\ell \dots b_1$ and $b_\ell \dots b_1 \sqsubseteq \mathrm{wrt}(w)$. From $\phi_4 \wedge \phi_5$ we infer that $w_1 \in \mathrm{redsup}_{\mathcal{I}}(b_\ell \dots b_1)$ holds for the prefix $w_1 = \mathrm{wrt}(w[1, q_1])$ of $\mathrm{wrt}(w)$.

For the converse implication, let $r_2 = b_\ell \dots b_1 \in A^\ell$ such that $\mathrm{rd}(w) \in A^* r_2$ and $w_1 \in \mathrm{redsup}_{\mathcal{I}}(r_2)$ holds for a prefix w_1 of $\mathrm{wrt}(w)$. Since there is no restriction to the order of x_i and y_i in embed_ℓ, the language $L(\mathsf{embed}_\ell)$ contains a word if, and only if, it contains all words with the same projections. Hence, we can assume that $\mathrm{rd}_2(w) = r_2$ and $w = \mathrm{nf}(w)$ holds. Let $p_1, \dots, p_\ell \in \mathrm{dom}(w)$ be the positions of the last ℓ read actions in w (in descending order) and let $\overline{b_1}, \dots, \overline{b_\ell} \in \overline{A}$ be the letters on these positions. Then by definition of $\mathrm{nf}(w)$ the positions $p_1 - 1, \dots, p_\ell - 1$ are labeled with $b_1, \dots, b_\ell$. Then it is easy to see, that

$$(\underline{w}, p_1, \dots, p_\ell, p_1 - 1, \dots, p_\ell - 1) \vDash \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5(x_1, \dots, x_\ell, y_1, \dots, y_\ell)$$

holds, i.e., we have $w \in L(\mathsf{embed}_\ell)$.

(2) Let $w \in L(\mathsf{overlap}_\ell)$. Then by (1) there is $r_2 = b_\ell \dots b_1 \in A^\ell$ such that $\mathrm{rd}(w) \in A^* r_2$ and $w_1 \in \mathrm{redsup}_{\mathcal{I}}(r_2)$ for a prefix w_1 of $\mathrm{wrt}(w)$. By satisfaction of ϕ_6 we obtain that each b_i is left of $\overline{b_i}$. By Theorem 4.7.8 we finally obtain $\mathrm{rd}_2(w) \in A^* r_2$.

Conversely, let $r_2 = b_\ell \dots b_1 \in A^\ell$ such that $\mathrm{rd}_2(w) \in A^* r_2$ and $w_1 \in \mathrm{redsup}_{\mathcal{I}}(r_2)$ for a prefix w_1 of $\mathrm{wrt}(w)$. Then by (1) we have $w \in L(\mathsf{embed}_\ell)$. Hence, we only have to check the satisfaction of ϕ_6. We prove this by induction on the minimal length n of a derivation $w \Rightarrow^n_{\mathfrak{R}_{\mathcal{I}}} \mathrm{nf}(w)$ where $\mathfrak{R}_{\mathcal{I}}$ is the semi-Thue system defined in Section 4.7 which orders the equations from Lemma 4.7.2 from left to right.

If $n = 0$ then we have $w = \mathrm{nf}(w)$. Set $k := |\mathrm{rd}_2(w)| \geq \ell$. Let $p_1, \dots, p_k \in \mathrm{dom}(w)$ be the positions of the last k read actions (in descending order) in w and let $\overline{b_1}, \dots, \overline{b_k} \in \overline{A}$ be the letters at these positions. Due to $\mathrm{wrt}_1(w) \in \mathrm{redsup}_{\mathcal{I}}(\mathrm{rd}_2(w)) = \mathrm{redsup}_{\mathcal{I}}(b_k \dots b_1)$ there is a factorization $v_k b_k v_{k-1} b_{k-1} \dots v_1 b_1 v_0 = \mathrm{wrt}(w)$ with $v_i \in (F \setminus \{b_i\})^*$ for any $1 \leq i \leq k$ and $v_0 = \mathrm{wrt}_2(w) \in A^*$. Then by $w = \mathrm{nf}(w)$ the positions $q_i := p_i - 1$ are labeled with b_i and, hence, we have $\mathrm{wrt}(w[q_{i+1} + 1, q_i]) = v_i b_i$ for each $1 \leq i \leq k$ (where $q_{k+1} := 0$). Since we also have $w_1 \in \mathrm{redsup}_{\mathcal{I}}(b_\ell \dots b_1)$ for a prefix w_1 of $\mathrm{wrt}(w)$ there is another factorization $v'_\ell b_\ell \dots v'_1 b_1 v'_0 = \mathrm{wrt}(w)$ with $v'_i \in (F \setminus \{b_i\})^*$ for any $1 \leq i \leq \ell$ and $v'_0 \in A^*$. Let $q'_\ell, \dots, q'_1 \in \mathrm{dom}(w)$ be the positions of $b_\ell, \dots, b_1$ with $\mathrm{wrt}(w[q'_{i+1} + 1, q'_i]) = v'_i b_i$ for each $1 \leq i \leq \ell$ (where $q'_{\ell+1} := 0$). Hence, by $\ell \leq k$ we can infer $q'_i \leq q_i = p_i - 1 < p_i$ for each $1 \leq i \leq \ell$. This finally implies

$$(\underline{w}, p_1, \dots, p_\ell, q'_1, \dots, q'_\ell) \vDash \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5 \wedge \phi_6(x_1, \dots, x_\ell, y_1, \dots, y_\ell)$$

i.e., we learn $w \in L(\mathsf{overlap}_\ell)$.

Now, assume $n > 0$. Then there is $w' \in \Sigma^*$ with $w \Rightarrow_{\mathfrak{R}_{\mathcal{I}}} w' \Rightarrow^{n-1}_{\mathfrak{R}_{\mathcal{I}}} \mathrm{nf}(w)$. By induction hypothesis we know $w' \in L(\mathsf{overlap}_\ell)$ and we have to show $w \in L(\mathsf{overlap}_\ell)$. We know that w' satisfies embed_ℓ. Since the application of any rule of $\mathfrak{R}_{\mathcal{I}}$ transposes only one write action with another read action, we also have $w \in L(\mathsf{embed}_\ell)$.

Let $p_1, \dots, p_\ell, q_1, \dots, q_\ell \in \mathrm{dom}(w') = \mathrm{dom}(w)$ be the positions in w' satisfying ϕ_1-ϕ_6. The transposition of the write and read action has two effects: the position of one read

action increases by one, i.e., at most one p_i increases. Additionally, the position of one write action decreases by one, i.e., at most one q_i decreases. This yields some positions $p'_1, \dots, p'_\ell, q'_1, \dots, q'_\ell \in \mathrm{dom}(w)$ satisfying ϕ_1-ϕ_5 in $\underline{w}$ and we have $p'_i \in \{p_i, p_i + 1\}$ and $q'_i \in \{q_i, q_i - 1\}$ for each $1 \leq i \leq \ell$. However, we also have

$$q'_i \leq q_i < p_i \leq p'_i$$

for each $1 \leq i \leq \ell$. In other words, the chosen positions satisfy ϕ_6 and, hence, we have $w \in L(\mathsf{overlap}_\ell)$. ◂

From this lemma we also obtain another characterization of the equivalence classes wrt. the behavioral equivalence $\equiv$. So, let $w \in \Sigma^*$ and $\ell \in \mathbb{N}$ be maximal such that $w \in L(\mathsf{overlap}_\ell)$. Moreover, let $k := \mathrm{obw}(w) < \infty$. Then we know that $w \in L(\mathsf{embed}_k \wedge \neg\mathsf{overlap}_k)$ holds. This implies, that a word $v \in \Sigma^*$ satisfies $v \equiv w$ whenever the last ℓ read actions of v appear right of their corresponding write actions and at least one of the last k read actions appears left of its corresponding write action. We will use this observation multiple times across this thesis. We can also infer the recognizability of the plq languages Ω_ℓ from this observation:

Lemma 6.4.14. *Let $\ell \in \mathbb{N}$. Then Ω_ℓ is aperiodic and, hence, recognizable.*

Proof. Since η is surjective, it suffices to show that $\eta^{-1}(\Omega_\ell)$ is aperiodic in Σ^*. By Observation 6.4.5 and Lemma 6.4.13 we have

$$\eta^{-1}(\Omega_\ell) = L\left(\bigwedge_{k=1}^{\ell} (\mathsf{embed}_k \to \mathsf{overlap}_k)\right).$$

Then by [MP71] $\eta^{-1}(\Omega_\ell)$ is aperiodic since the given formula is contained in $\mathsf{FO}[\lambda]$. ◂

Finally, this lemma implies the implication "(iii)⇒(i)" in Theorem 6.4.6 and, hence, finishes the proof of Theorem 6.4.6.

Proposition 6.4.15. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ and $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ be a Boolean combination of plq languages of the form $\mathrm{wrt}^{-1}(R)$, $\mathrm{rd}^{-1}(R)$, and Ω_ℓ for regular languages $R \subseteq A^*$ and numbers $\ell \in \mathbb{N}$. Then T is recognizable.*

Proof. Since $\mathrm{wrt}^{-1}(R)$ and $\mathrm{rd}^{-1}(R)$ are recognizable for any regular language $R \subseteq A^*$ and since Ω_ℓ is recognizable for any $\ell \in \mathbb{N}$ due to Lemma 6.4.14, the language T is recognizable by closure properties of the class of recognizable languages in $\mathbb{T}(\mathcal{Q}_\mathcal{L})$. ◂

6.4.2 From Recognizability to Q-Rationality

Now, we are able to prove the equivalences from our main theorem in this section (Theorem 6.4.1). In this subsection we prove that each recognizable language of the plq monoid is

q-rational. To this end, we first have to define this notion which is a restriction to the classical rational expressions. We need this restriction since the classes of rational and recognizable plq languages do not coincide (cf. Theorem 5.4.5). Note that this is in contrast to Kleene's Theorem [Kle51] stating that the rational and recognizable word languages coincide. Though, we can use Ochmański's approach from [Och85] to generate the recognizable languages. Concretely, we restrict the iteration and the concatenation of the plq monoid in an appropriate way. Unfortunately, we still cannot generate all recognizable plq languages by union, restricted product, and restricted iteration (we will see an example later in this subsection). Hence, we have to add another, non-monotonic[xiii] operation to our expressions: the complement operation. We call the plq languages generated by those operations *q-rational* and prove that these are exactly the recognizable languages in the plq monoid.

At first, we prove that the class of recognizable plq languages is neither closed under iteration nor it is closed under concatenation:

Proposition 6.4.16. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$. Then the following statements hold:*

(1) There is a recognizable language $T \subseteq \mathbb{T}(Q_{\mathcal{L}})$ such that T^* is not recognizable in $\mathbb{T}(Q_{\mathcal{L}})$.

(2) There are recognizable languages $S, T \subseteq \mathbb{T}(Q_{\mathcal{L}})$ such that $S \cdot T$ is not recognizable in $\mathbb{T}(Q_{\mathcal{L}})$.

Proof. Let $a \in A$ be a letter.

Towards the former statement set $T := \{[\![a\overline{a}]\!]\}$ which is recognizable since $\eta^{-1}(T)$ is finite. Then we have

$$\eta^{-1}(T^*) \cap A^*\overline{A}^* = \{a^n\overline{a}^n \mid n \in \mathbb{N}\}$$

due to equation (4) from Lemma 4.7.2. This language is not regular. Hence, by closure properties of the class of regular languages $\eta^{-1}(T^*)$ is not recognizable and thus T^* is not recognizable.

Towards the latter statement set $S := \{[\![a]\!]\}^*$ and $T := \{[\![\overline{a}]\!]\}^*$. Since $\eta^{-1}(S) = a^*$ and $\eta^{-1}(T) = \overline{a}^*$ holds, both plq languages are recognizable in $\mathbb{T}(Q_{\mathcal{L}})$. Then we have

$$\eta^{-1}(S \cdot T) \cap \overline{A}A^*\overline{A}^* = \{\overline{a}a^m\overline{a}^n \mid m, n \in \mathbb{N}, m \leq n\}$$

due to Lemma 4.7.2(4). This language is not regular and, hence, $S \cdot T$ is not recognizable. ◂

Note that Proposition 6.4.16(1) is a very similar situation as in trace monoids. Here, Ochmański proved in [Och85] that there are recognizable trace languages such that their iteration is not recognizable anymore. However, the iteration of connected, recognizable trace languages always is recognizable. Hence, this led to the definition of the so-called c-rational expressions.

In the plq monoid, we also have to restrict concatenation due to Proposition 6.4.16(2). According to these restrictions we will define now the so-called *q-rational* languages in the

[xiii] Let $\leq$ be a partial ordering on a set S. A function $f\colon S \to S$ is *monotonic* if for each $x \leq y$ we also have $f(x) \leq f(y)$.

plq monoid. Afterwards we prove that this is a suitable restriction of rationality to describe exactly the recognizable plq languages.

Let $\mathcal{L} = (F, U)$ be a lossiness alphabet. At first, we say that a $\mathbb{T}(\mathcal{Q}_\mathcal{L})$-language is q^+-*rational* if it can be obtained by the following rules:

(1^+) $\mathrm{wrt}^{-1}(\varepsilon)$, $\mathrm{wrt}^{-1}(\varnothing) = \varnothing$, and $\mathrm{wrt}^{-1}(a)$ are q$^+$-rational for any $a \in A$.

(2^+) if $S, T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ are q$^+$-rational then $S \cup T$, $S \cdot T$, and S^* are q$^+$-rational.

Similarly, by replacing wrt^{-1} by rd^{-1} in the rules above, we obtain the class of q^-*-rational* languages in $\mathbb{T}(\mathcal{Q}_\mathcal{L})$.

Observation 6.4.17. *Let $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$. Then the following statements hold:*

(1) T is q^+-rational if, and only if, there is a regular language $R \subseteq A^*$ with $T = \mathrm{wrt}^{-1}(R)$.

(2) T is q^--rational if, and only if, there is a regular language $R \subseteq A^*$ with $T = \mathrm{rd}^{-1}(R)$.

Proof idea. Both equivalences hold since $\mathrm{wrt}^{-1}, \mathrm{rd}^{-1} \colon 2^{A^*} \to 2^{\mathbb{T}(\mathcal{Q}_\mathcal{L})}$ are homomorphisms with respect to union, concatenation, and iteration. ◂

Finally, a $\mathbb{T}(\mathcal{Q}_\mathcal{L})$-language is *q-rational* if it can be constructed by the following rules:

(1) if $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ is q$^+$- or q$^-$-rational then it is also q-rational

(2) if $S, T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ are q-rational then $S \cup T$ and $\mathbb{T}(\mathcal{Q}_\mathcal{L}) \setminus S$ are q-rational

(3) if $S \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ is q$^+$-rational and $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ is q$^-$-rational such that $\mathrm{rd}(T)$ is finite (i.e., T is obtained without usage of the $*$-operator) then $S \cdot \mathbb{T}(\mathcal{Q}_\mathcal{L}) \cdot T$ is q-rational.

Example 6.4.18. Let $T = \{t \in \mathbb{T}(\mathcal{Q}_\mathcal{L}) \mid \mathrm{wrt}(t) \in (ab)^*, \mathrm{rd}(t) = b\}$. Then T is q-rational since we have

$$T = \mathrm{rd}^{-1}(b) \cap (\mathrm{wrt}^{-1}(a) \cdot \mathrm{wrt}^{-1}(b))^* .$$

Note that the class of q-rational languages also is closed under intersection due to Rule (2), i.e., this class is a Boolean algebra. The plq language T also is recognizable since $\eta^{-1}(T) = (ab)^* \sqcup\!\sqcup \overline{b}^*$ is regular. However, we cannot construct T from the rules above without usage of complement (or intersection). ⌟

Remark 6.4.19. Recall the classical definition of rational languages. The class of these languages is the closure of the finite languages under union, concatenation, and iteration. All of these operations are monotonic. For q-rationality we additionally need the closure under complement which is not monotonic. Until now it is still an open question whether there is another characterization of the recognizable plq languages using monotonic operations, only. ⌟

At first sight, the choice of Rule (3) seems to be some kind of random. The following example shows that we can remove neither the factor "$\mathbb{T}(\mathcal{Q}_\mathcal{L})$", which appears as separator in this product, nor the finiteness of $\mathrm{rd}(T)$. Additionally, we cannot simply remove this rule since the recognizable language $\{[\![a\overline{a}]\!]\}$ cannot be built by application of the Rules (1) and (2), only.

Example 6.4.20. Let $a, b \in A$ be distinct letters. Then the language $\mathrm{wrt}^{-1}((ab)^*a) \cdot \mathrm{rd}^{-1}(\varepsilon)$ is not recognizable since

$$\eta^{-1}\left(\mathrm{wrt}^{-1}((ab)^*a) \cdot \mathrm{rd}^{-1}(\varepsilon)\right) \cap (ab)^*(\overline{a}\overline{b})^*$$

contains exactly those words $(ab)^m(\overline{a}\overline{b})^n$ with $m > n$, which is not regular.

Now, let $a, b, c \in A$ be distinct letters with $a, c \in U$. Then the plq language

$$\mathrm{wrt}^{-1}(aA^*c) \cdot \mathbb{T}(\mathcal{Q}_{\mathcal{L}}) \cdot \mathrm{rd}^{-1}(aA^*c)$$

is not recognizable since

$$L := \eta^{-1}\left(\mathrm{wrt}^{-1}(aA^*c) \cdot \mathbb{T}(\mathcal{Q}_{\mathcal{L}}) \cdot \mathrm{rd}^{-1}(aA^*c)\right) \cap \overline{ab^*c}ab^*c$$

contains exactly those words $\overline{ab^mc}ab^nc$ satisfying $m \neq n$ if $b \in U$ or $m > n$ if $b \in F$ holds. In both cases L is not regular. ⌟

Now we can prove the implication "(A)⇒(B)" in Theorem 6.4.1. To do this, we utilize Theorem 6.4.6(iii). Concretely, we understand a recognizable plq language as a Boolean combination of languages of the form $\mathrm{wrt}^{-1}(R)$, $\mathrm{rd}^{-1}(R)$, and Ω_ℓ for regular languages $R \subseteq A^*$ and numbers $\ell \in \mathbb{N}$. Then we prove q-rationality by induction on the syntax tree of such expression. The most complicated case in this proof is to show that Ω_ℓ is q-rational. For this proof we need the following lemma:

Lemma 6.4.21. *Let $\ell \in \mathbb{N}$, $t \in \mathbb{T}(\mathcal{Q}_{\mathcal{L}})$, and $r_2 = a_1 \dots a_\ell$ with $a_1, \dots, a_\ell \in A$. Then we have $\mathrm{rd}_2(t) \in A^*r_2$ and $w_1 \in \mathrm{redsup}_{\mathcal{L}}(r_2)$ for a prefix w_1 of $\mathrm{wrt}(t)$ if, and only if,*

$$t \in \mathrm{wrt}^{-1}\left(\prod_{i=1}^{\ell} F^*a_i\right) \cdot \mathbb{T}(\mathcal{Q}_{\mathcal{L}}) \cdot \mathrm{rd}^{-1}(r_2)\ .$$

Proof. First, assume $\mathrm{rd}_2(t) \in A^*r_2$ and $w_1 \in \mathrm{redsup}_{\mathcal{L}}(r_2)$ for a prefix w_1 of $\mathrm{wrt}(t)$. From Corollary 4.7.15 we know $t = [\![\overline{\mathrm{rd}_1(t)}\mathrm{wrt}(t)\overline{\mathrm{rd}_2(t)}]\!]$. Then by assumption we obtain

$$t = \left[\!\!\left[\overline{\mathrm{rd}_1(t)}\mathrm{wrt}(t)\overline{\mathrm{rd}_2(t)}\right]\!\!\right] \in \mathrm{wrt}^{-1}\left(\prod_{i=1}^{\ell} F^*a_i\right) \cdot \mathbb{T}(\mathcal{Q}_{\mathcal{L}}) \cdot \mathrm{rd}^{-1}(r_2)\ .$$

Now, assume $t \in \mathrm{wrt}^{-1}\left(\prod_{i=1}^{\ell} F^*a_i\right) \cdot \mathbb{T}(\mathcal{Q}_{\mathcal{L}}) \cdot \mathrm{rd}^{-1}(r_2)$. Then we have $w_1 \in \mathrm{redsup}_{\mathcal{L}}(r_2)$ for a prefix w_1 of $\mathrm{wrt}(t)$ and $\mathrm{rd}(t) \in A^*r_2$. Furthermore, there are $w_1, w_2, w_3 \in \Sigma^*$ with $t = [\![w_1w_2w_3]\!]$, $\mathrm{wrt}(w_1) \in \prod_{i=1}^{\ell} F^*a_i$, and $\mathrm{rd}(w_3) = r_2$. Then the letters $\overline{a_1}, \dots, \overline{a_\ell}$ appear to the right of $a_1, \dots, a_\ell$ in $w_1w_2w_3$, i.e., $w_1w_2w_3 \in L(\mathsf{overlap}_\ell)$. Hence, by Lemma 6.4.13(2) we obtain $\mathrm{rd}_2(t) = \mathrm{rd}_2(w_1w_2w_3) \in A^*r_2$. ◂

Finally, we can state the following implication:

Proposition 6.4.22. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ and $T \subseteq \mathbb{T}(Q_{\mathcal{L}})$ be recognizable. Then T is q-rational.*

Proof. By Theorem 6.4.6(iii) T is a Boolean combination of plq languages of the form $\mathrm{wrt}^{-1}(R)$, $\mathrm{rd}^{-1}(R)$, and Ω_ℓ for regular languages $R \subseteq A^*$ and numbers $\ell \in \mathbb{N}$. We prove the claim by induction on the syntax tree.

At first, assume $T = \mathrm{wrt}^{-1}(R)$ for a regular language $R \subseteq A^*$. Then T is q^+-rational by Observation 6.4.17(1) and, hence, q-rational. Similarly, $T = \mathrm{rd}^{-1}(R)$ is q^--rational and, therefore, q-rational for any regular language $R \subseteq A^*$.

Next, suppose $T = \Omega_\ell$ for a number $\ell \in \mathbb{N}$. Then by Observation 6.4.5 and Lemma 6.4.21 we have

$$\Omega_\ell = \bigcap_{r_2 \in A^{\leq \ell}} \Big(\mathbb{T}(Q_{\mathcal{L}}) \setminus \big(\mathrm{wrt}^{-1}(W_{r_2} A^*) \cap \mathrm{rd}^{-1}(A^* r_2) \big) \cup \mathrm{wrt}^{-1}(W_{r_2}) \cdot \mathbb{T}(Q_{\mathcal{L}}) \cdot \mathrm{rd}^{-1}(r_2) \Big),$$

where $W_{r_2} = \prod_{i=1}^{k} F^* a_i$ with $r_2 = a_1 \dots a_k$. Since the plq languages $\mathrm{wrt}^{-1}(W_{r_2} A^*)$, $\mathrm{rd}^{-1}(A^* r_2)$, $\mathrm{wrt}^{-1}(W_{r_2})$, and $\mathrm{rd}^{-1}(r_2)$ are q-rational by the first case of this proof, Ω_ℓ is q-rational due to Rules (2) and (3).

Finally, assume $T = S_1 \cup S_2$, $T = S_1 \cap S_2$, or $T = \mathbb{T}(Q_{\mathcal{L}}) \setminus S_1$. Then by induction hypothesis S_1 and S_2 are q-rational. Hence, using Rule (2) T is q-rational as well. ◂

6.4.3 From Q-Rationality to Logical Definability

The second implication from Theorem 6.4.1 states that each q-rational plq language is definable in a special monadic second-order logic which we call $\mathsf{MSO}[o]$. So, the aim of this subsection is to derive such signature from Büchi's word-logic λ and corresponding structures $\mathcal{S}(w)$ for each word $w \in \Sigma^*$ such that we have the following properties:

(1) For two action sequences $v, w \in \Sigma^*$ we have $\mathcal{S}(v) \cong \mathcal{S}(w)$ if, and only if, $v \equiv w$ holds. By definition of $\equiv$ this is the case if, and only if, v and w induce the same transformation $[\![v]\!] = [\![w]\!]$.

(2) The monadic second-order logic on these structures describes exactly the recognizable properties in the plq monoid. To this end, we have to exhibit the knowledge from the preceding subsections.

To this end, we have to revisit the rules from the semi-Thue system $\mathfrak{R}_{\mathcal{L}}$. First, we can observe that the application of any rule of $\mathfrak{R}_{\mathcal{L}}$ to the word w does not change the projections $\mathrm{wrt}(w)$ and $\mathrm{rd}(w)$. For example, if the i^{th} read action in w is right of the j^{th} read action (i.e., $i < j$), then this also holds for any action sequence satisfying $v \equiv w$. In particular, the i^{th} read action in w agrees with the i^{th} read action in v in this case.

So, similar to the word models $\underline{w}$, the universe of our new structure $\mathcal{S}(w)$ is the set of positions $\mathrm{dom}(w)$ in w. Additionally, we need a partial ordering R satisfying the following property: $p \, R \, q$ if, and only if: (1) p is the position of the i^{th} write or read action, (2) q is the position of the j^{th} write or read action, and (3) for each $v \equiv w$ the i^{th} write / read action in v appears to the left of the j^{th} write / read action in v. Unfortunately, MSO-formulas over these

new structures are able to describe non-recognizable properties as the following remark shows:

Remark 6.4.23. Consider the FO-formula $\psi := \exists x\colon \Lambda_A(x) \wedge \forall y\colon \Lambda_{\overline{A}}(y) \to x\, R\, y$, i.e., $[\![w]\!]$ satisfies ψ if, and only if, for each $v \equiv w$ there is a write action appearing left of all read actions. In other words, this is the language of all words $w \in \Sigma^*$ with $\mathrm{rd}_1(w) = \varepsilon$. Then we can see

$$\{w \in \Sigma^* \mid \mathcal{S}(w) \vDash \psi\} \cap \overline{a}^* a^* \overline{a}^* = \{a^m \overline{a}^n \mid m \geq n\}$$

which is no regular language. Hence, the language of all transformations $t \in \mathbb{T}(Q_{\mathcal{L}})$ satisfying ψ is not recognizable. ⌟

So, we have to weaken our structures $\mathcal{S}(w)$. A possible approach is to split R into the following relations: $<_+$ which is the restriction of R to the positions of write actions, $<_-$ which is the restriction to the positions of read actions, and R_{rw} which is the restriction of R to pairs (p, q) where p is the position of a read action and q is the position of a write action. In other words, we remove those pairs (p, q) from R where p is the position of a write action and q is one of a read action. Unfortunately, those weakened structures are still too expressive:

Remark 6.4.24. Consider the FO-formula

$$\phi := \exists x, y\colon \begin{pmatrix} \Lambda_A(x) \wedge \forall z\colon (\Lambda_A(z) \to z \leq_+ x) \wedge \\ \Lambda_{\overline{A}}(y) \wedge \forall z\colon (\Lambda_{\overline{A}}(z) \to y \leq_- z) \wedge \neg y\, R_{\mathrm{rw}}\, x \end{pmatrix},$$

i.e., $[\![w]\!]$ satisfies ϕ if, and only if, there is $v \equiv w$ in which the first read action appears to the right of the last write action. Then we have

$$\{w \in \Sigma^* \mid \mathcal{S}(w) \vDash \phi\} \cap \overline{a}^* a^* \overline{a}^* = \{\overline{a}^k a^\ell \overline{a}^m \mid k = 0 \text{ or } m \geq \ell\}.$$

Since the right-hand side of the equation is not regular, the language of all transformations $t \in \mathbb{T}(Q_{\mathcal{L}})$ satisfying ϕ is not recognizable either. ⌟

Another way to weaken the expressiveness of these structures is the following: we split R_{rw} into relations $R_{\mathrm{rw},\ell}$ where $\ell > 0$ is a positive integer. This relation $R_{\mathrm{rw},\ell}$ contains exactly those tuples (p, q) from R_{rw} where p is the position of the ℓ^{th} last read action. Note that we have

$$R_{\mathrm{rw}} = \bigcup_{\ell > 0} R_{\mathrm{rw},\ell}\,.$$

We can see that for a given word $w \in \Sigma^*$ and a number $\ell > 0$ all tuples in the relation $R_{\mathrm{rw},\ell}$ agree in their first component. Hence, we are able to project these relations to their second component yielding the sets $P_\ell := \{q \in \mathrm{dom}(w) \mid \exists p \in \mathrm{dom}(w)\colon p\, R_{\mathrm{rw},\ell}\, q\}$.

All in all, we can define the (infinite) signature o[xiv] consisting of the binary symbols $<_+$ and $<_-$ and the unary symbols P_ℓ and Λ_α for each $\ell > 0$ and $\alpha \in \Sigma$. Let $w = \alpha_1 \dots \alpha_n$ with $\alpha_1, \dots, \alpha_n \in \Sigma$. The *queue model* of w is the relational o-structure

$$\widetilde{w} := (\mathrm{dom}(w), <_+^w, <_-^w, (P_\ell^w)_{\ell > 0}, (\Lambda_\alpha^w)_{\alpha \in \Sigma})$$

[xiv] The name of the signature "o" stems from the greek phrase "$o\upsilon\rho\acute{\alpha}$" which translates to "queue".

where $\mathrm{dom}(w) := \{1, \ldots, n\}$ is the set of positions in w, $\Lambda_\alpha^w := \{p \in \mathrm{dom}(w) \mid \alpha_p = \alpha\}$ are the labelings of the positions, $<_+^w$ and $<_-^w$ are the natural orderings on $\Lambda_A^w := \bigcup_{a\in A} \Lambda_a^w$ and $\Lambda_{\overline{A}}^w$, respectively, and

$$P_\ell^w := \{p \in \Lambda_A^w \mid \forall v_1, v_2 \in \Sigma^* : w \equiv v_1 v_2, \mathrm{wrt}(v_1) = \mathrm{wrt}(w[1,p]) \Rightarrow |\mathrm{rd}(v_2)| < \ell\} \ .$$

In other words, we have $p \in P_\ell^w$ if, and only if, $\alpha_p \in A$ and the ℓ^{th} last read action in w is to the left of α_p in any equivalently behaving word $v \equiv w$. This is conform to the approaches known from [Büc60, Ebi95] since the relations $<_+^w$, $<_-^w$, and P_ℓ^w specify which letter have to appear to the left of another one in any word equivalent to w.

Before we show that the $\mathsf{MSO}[o]$-formulas describe exactly the recognizable plq languages, we first show several basic properties of the queue model $\widetilde{w}$ of an action sequence $w \in \Sigma^*$. First, we show that $\widetilde{w}$ identifies the equivalence class $[w]_\equiv$:

Lemma 6.4.25. *Let $v, w \in \Sigma^*$. Then we have $v \equiv w$ if, and only if, $\widetilde{v} \cong \widetilde{w}$.*

Proof. At first, we assume $v \equiv w$. Then by Theorem 4.7.8 there is a permutation σ of $\mathrm{dom}(v) = \mathrm{dom}(w)$ such that σ is compatible with $<_+$, $<_-$, and Λ_α for any $\alpha \in \Sigma$. Additionally, by definition σ is compatible with P_ℓ for any $\ell > 0$. Hence, σ is an isomorphism from $\widetilde{v}$ into $\widetilde{w}$, i.e., $\widetilde{v} \cong \widetilde{w}$.

Now assume $v \not\equiv w$. If $\mathrm{wrt}(v) \neq \mathrm{wrt}(w)$ or $\mathrm{rd}(v) \neq \mathrm{rd}(w)$ then we know $\underline{\mathrm{wrt}(v)} \not\equiv \underline{\mathrm{wrt}(w)}$ or $\underline{\mathrm{rd}(v)} \not\equiv \underline{\mathrm{rd}(w)}$. Note that these word models are substructures of the queue models $\widetilde{v}$ resp. $\widetilde{w}$. Hence, any possible bijection from $\mathrm{dom}(v)$ into $\mathrm{dom}(w)$ (if there is any) cannot be compatible with $<_+$, $<_-$, and Λ_α at the same time. This implies $\widetilde{v} \not\cong \widetilde{w}$. So, we can assume $\mathrm{wrt}(v) = \mathrm{wrt}(w)$ and $\mathrm{rd}(v) = \mathrm{rd}(w)$ from now on. Let σ be the uniquely determined permutation of $\mathrm{dom}(v) = \mathrm{dom}(w)$ which is compatible to $<_+$, $<_-$, and Λ_α for any $\alpha \in \Sigma$.

By Theorem 4.7.8 we infer from $v \not\equiv w$ that $\mathrm{rd}_2(v) \neq \mathrm{rd}_2(w)$ holds. Since both words are suffixes of $\mathrm{rd}(v) = \mathrm{rd}(w)$ we may assume that $\mathrm{rd}_2(w)$ is a proper suffix of $\mathrm{rd}_2(v)$, i.e., $|\mathrm{rd}_2(v)| > |\mathrm{rd}_2(w)|$. Note that we have $\mathrm{wrt}(v) = \mathrm{wrt}(w) \neq \varepsilon$, since otherwise we would have $\mathrm{rd}_2(v) = \mathrm{rd}_2(w) = \varepsilon$ by definition of $\mathrm{redsup}_{\mathcal{Q}}$.

Let $p \in \mathrm{dom}(v)$ be the position of the last write action in v. From Corollary 4.7.15 we know $v \equiv \overline{\mathrm{rd}_1(v)}\mathrm{wrt}(v)\overline{\mathrm{rd}_2(v)}$ and, hence, $p \notin P^v_{|\mathrm{rd}_2(v)|}$. Now, let $w_1, w_2 \in \Sigma^*$ with $w \equiv w_1 w_2$ and $\mathrm{wrt}(w_1) = \mathrm{wrt}(w[1, \sigma(p)]) = \mathrm{wrt}(w)$ (note that $\sigma(p)$ is the position of the last write action in w). Suppose $|\mathrm{rd}(w_2)| \geq |\mathrm{rd}_2(v)|$. Then we have $w_1 w_2 \in L(\mathsf{overlap}_{|\mathrm{rd}_2(v)|})$ (note that $x \in \mathrm{redsup}_{\mathcal{Q}}(\mathrm{rd}_2(v))$ holds for a prefix x of $\mathrm{wrt}(w) = \mathrm{wrt}(v)$) implying $|\mathrm{rd}_2(w)| = |\mathrm{rd}_2(w_1 w_2)| \geq |\mathrm{rd}_2(v)|$ by Lemma 6.4.13(2) - contracting to our assumption $|\mathrm{rd}_2(w)| < |\mathrm{rd}_2(v)|$. Hence, we infer $|\mathrm{rd}(w_2)| < |\mathrm{rd}_2(v)|$. Then we have $\sigma(p) \in P^w_{|\mathrm{rd}_2(v)|}$ and thus σ is no isomorphism from $\widetilde{v}$ into $\widetilde{w}$. Since σ is unique, we have $\widetilde{v} \not\cong \widetilde{w}$. ◂

Due to Lemma 6.4.25 we are able to define the *queue model* of a transformation $t \in \mathbb{T}(Q_{\mathcal{Q}})$ by $\widetilde{t} := \widetilde{\mathrm{nf}(t)}$.

By definition, the signature o is infinite. However, we can represent these structures finitely, since we can split $\mathbb{N} \setminus \{0\}$ into at most three intervals I_1, I_2, and I_3 such that the sets P_ℓ^w and P_k^w coincide if, and only if, ℓ and k belong to the same interval I_i. These intervals are

closely related to $\mathrm{rd}(w)$, $\mathrm{wrt}(w)$, and $\mathrm{obw}(w)$: if $|\mathrm{wrt}(w)| = 0$ we have $w \in \overline{A}^*$ and, hence, $P_\ell^w = \varnothing$ for each $\ell > 0$. The case $|\mathrm{wrt}(w)| > 0$ is considered in the following observation:

Observation 6.4.26. *Let $w \in \Sigma^*$ with $|\mathrm{wrt}(w)| > 0$. Then the following statements hold:*

(1) For each $1 \leq \ell < \min\{\mathrm{obw}(w), |\mathrm{rd}(w)| + 1\}$ we have $P_\ell^w = \varnothing$.

(2) For each $\mathrm{obw}(w) \leq \ell \leq |\mathrm{rd}(w)|$ we have $P_\ell^w = P_{\mathrm{obw}(w)}^w \neq \varnothing$.

(3) For each $\ell > |\mathrm{rd}(w)|$ we have $P_\ell^w = \Lambda_A^w$.

Proof. (1) Since $\ell < \mathrm{obw}(w)$ holds, we have $[\![w]\!] \in \Omega_\ell$. Let $r_1, r_2 \in A^*$ with $\mathrm{rd}(w) = r_1 r_2$ and $|r_2| = \ell$. Then, by Lemma 6.4.9 there is $v \in \Sigma^*$ such that $w \equiv v\overline{r_2}$ holds. Consequently, any write action in w can be followed by $\geq \ell$ read actions, i.e., $P_\ell^w = \varnothing$.

(2) First, we show $p := \max \Lambda_A^w \in P_{\mathrm{obw}(w)}^w \neq \varnothing$ (i.e., the last write position in w is in this set). Let $v_1, v_2 \in \Sigma^*$ with $w \equiv v_1 v_2$ and $\mathrm{wrt}(v_1) = \mathrm{wrt}(w[1, p]) = \mathrm{wrt}(w)$. We show $|\mathrm{rd}(v_2)| < \mathrm{obw}(w)$. Since we have $|\mathrm{rd}_2(w)| < \mathrm{obw}(w) \leq |\mathrm{rd}(w)|$ we obtain $w \in L(\mathsf{embed}_{\mathrm{obw}(w)})$ and $w \notin L(\mathsf{overlap}_{\mathrm{obw}(w)})$. From Lemma 6.4.13, $w \equiv v_1 v_2$, and $w \notin L(\mathsf{overlap}_{\mathrm{obw}(w)})$ we also obtain $v_1 v_2 \notin L(\mathsf{overlap}_{\mathrm{obw}(w)})$. Hence, the satisfaction of ϕ_6 implies that $|\mathrm{rd}(v_2)| < \mathrm{obw}(w)$ and, therefore, $p \in P_{\mathrm{obw}(w)}^w$.

Next, we prove $P_\ell^w = P_{\mathrm{obw}(w)}^w$. By definition, the inclusion "$\supseteq$" is trivial. We prove the converse inclusion "$\subseteq$" by induction on ℓ. If $\ell = \mathrm{obw}(w)$, we are done. So, assume $\ell > \mathrm{obw}(w)$. Let $p \in \Lambda_A^w \setminus P_{\ell-1}^w = \Lambda_A^w \setminus P_{\mathrm{obw}(w)}^w$. Then there are $v_1, v_2 \in \Sigma^*$ with $w = v_1 v_2$, $\mathrm{wrt}(v_1) = \mathrm{wrt}(w[1, p])$, and $|\mathrm{rd}(v_2)| \geq \ell - 1$. We are done if $|\mathrm{rd}(v_2)| \geq \ell$ holds. Hence, we assume $|\mathrm{rd}(v_2)| = \ell - 1$ from now on.

Set $r_1 := \mathrm{rd}(v_1)$ and $r_2 := \mathrm{rd}(v_2)$, i.e., $\mathrm{rd}(w) = \mathrm{rd}(v_1 v_2) = r_1 r_2$. Additionally, let $r_1 = r_1' a$ with $a \in A$ and let $v_1' \in \Sigma^*$ be the word arising from v_1 by removing the right-most occurrence of $\overline{a}$.

Suppose $w_1 \notin \mathrm{redsup}_{\mathcal{L}}(a r_2)$ for any prefix w_1 of $\mathrm{wrt}(w)$. Using the rules from the semi-Thue system $\mathfrak{R}_{\mathcal{L}}$, we can move the letter $\overline{a}$ in $v_1 v_2$ to the right-hand side of the write action on position p in w. Then we have $v_1' \overline{a} v_2 \equiv v_1 v_2 \equiv w$ (since $\mathrm{rd}_2(v_1' \overline{a} v_2) = \mathrm{rd}_2(v_1 v_2)$). Additionally, we know $\mathrm{wrt}(v_1') = \mathrm{wrt}(v_1) = \mathrm{wrt}(w[1, p])$ and $|\mathrm{rd}(\overline{a} v_2)| = \ell$. This finally implies $p \in \Lambda_A^w \setminus P_\ell^w$.

Now, assume $w_1 \in \mathrm{redsup}_{\mathcal{L}}(a r_2)$ for a prefix w_1 of $\mathrm{wrt}(w)$. We know

$$|\mathrm{rd}_2(v_1 v_2)| = |\mathrm{rd}_2(w)| < \mathrm{obw}(w) < \ell\,.$$

By definition of $\mathrm{obw}(w)$, there is $u \in A^{\mathrm{obw}(w)}$ with $\mathrm{rd}(v_1 v_2) = \mathrm{rd}(w) \in A^* u$, $w_1' \in \mathrm{redsup}_{\mathcal{L}}(u)$ for a prefix w_1' of $\mathrm{wrt}(w)$, and $|\mathrm{rd}_2(v_1 v_2)| < |u| = \mathrm{obw}(w)$. Let $r_2 = b_{\ell-1} \dots b_1$ where $b_1, \dots, b_{\ell-1} \in A$ are letters and let $p_1, \dots, p_{\mathrm{obw}(w)} \in \mathrm{dom}(w) = \mathrm{dom}(v_1 v_2)$ be the positions of $\overline{b_1}, \dots, \overline{b_{\mathrm{obw}(w)}}$ in $v_1 v_2$. Additionally, let $q_1, \dots, q_{\mathrm{obw}(w)} \in \mathrm{dom}(w)$ be the positions of the corresponding write actions b_i in $v_1 v_2$, i.e., $\mathrm{wrt}(v_1 v_2[q_{i+1} + 1, q_i - 1]) \in (F \setminus \{b_i\})^*$ holds for each $1 \leq i \leq \mathrm{obw}(w)$ (where $q_{\mathrm{obw}(w)+1} := 0$). By Lemma 6.4.13 there is $1 \leq i \leq \mathrm{obw}(w)$ such that $q_i > p_i$.

Consider the positions $p_1', \dots, p_\ell' \in \mathrm{dom}(w)$ of the ℓ right-most read actions (in descending order) and their corresponding write actions $q_1', \dots, q_\ell' \in \mathrm{dom}(w)$ in $v_1 v_2$.

Then we have $q_i' \geq q_i > p_i = p_i'$ implying, by application of Lemma 6.4.13, $|\mathrm{rd}_2(v_1'\overline{a}v_2)| < \ell = |ar_2|$. Hence, we can infer $\mathrm{rd}_2(v_1v_2) = \mathrm{rd}_2(v_1'\overline{a}v_2)$. But this finally implies $w \equiv v_1v_2 \equiv v_1'\overline{a}v_2$ and, therefore, $p \in \Lambda_A^w \setminus P_\ell^w$.

(3) Since w contains $< \ell$ read actions, no write action can ever be followed by ℓ read actions. Consequently, we have $P_\ell^w = \Lambda_A^w$. ◂

Let $\phi \in \mathsf{MSO}[o]$ be a sentence. The language of transformations defined by ϕ is $T(\phi) := \{t \in \mathbb{T}(\mathcal{Q}_\mathcal{L}) \mid \widetilde{t} \vDash \phi\}$. We say that $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ is $\mathsf{MSO}[o]$-*definable* ($\mathsf{FO}[o]$-*definable*) if there is a sentence $\phi \in \mathsf{MSO}[o]$ ($\phi \in \mathsf{FO}[o]$, respectively) with $T = T(\phi)$.

Remark 6.4.27. The sets P_ℓ^w also conform with the special product in the definition of q-rational languages. In particular, we have

$$T(\exists x\colon \neg P_\ell(x) \wedge \Lambda_A(x)) = \mathrm{wrt}^{-1}(A^+) \cdot \mathbb{T}(\mathcal{Q}_\mathcal{L}) \cdot \mathrm{rd}^{-1}\big(A^\ell\big)\ . \qquad \lrcorner$$

Finally, we can state the following implication:

Proposition 6.4.28. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ and $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ be q-rational. Then T is $\mathsf{MSO}[o]$-definable.*

Proof. We prove this by induction on the syntax tree of a q-rational expression generating the plq language T.

If T is q^+-rational then we have $T = \mathrm{wrt}^{-1}(R)$ for a regular $R \subseteq A^*$ by Observation 6.4.17. By [Büc60] there is an $\mathsf{MSO}[\lambda]$-formula ϕ with $L(\phi) = R$. Then by replacing all occurrences of $<$ in ϕ by $<_+$ we obtain an $\mathsf{MSO}[o]$-formula ϕ' with $T(\phi'\restriction_{\Lambda_A(x)}) = \mathrm{wrt}^{-1}(L(\phi)) = T$ (recall that $\psi\restriction_\xi$ restricts the quantifiers of ψ to the elements satisfying ξ).

Similarly, we can prove that T is $\mathsf{MSO}[o]$-definable if T is q^--rational (here, we replace $<$ by $<_-$ and restrict the quantifiers to $\Lambda_{\overline{A}}$).

If $T = S_1 \cup S_2$ or $T = \mathbb{T}(\mathcal{Q}_\mathcal{L}) \setminus S_1$ where S_1, S_2 are q-rational, there are $\phi_1, \phi_2 \in \mathsf{MSO}[o]$ with $T(\phi_1) = S_1$ and $T(\phi_2) = S_2$ by induction hypothesis. Then we have $T = T(\phi_1 \vee \phi_2)$ and $T = T(\neg\phi_1)$, respectively.

Finally, let $T = \mathrm{wrt}^{-1}(R_1) \cdot \mathbb{T}(\mathcal{Q}_\mathcal{L}) \cdot \mathrm{rd}^{-1}(R_2)$ where $R_1 \subseteq A^*$ is regular and $R_2 \subseteq A^*$ is finite. W.l.o.g. we can assume that $R_2 = \{w\}$ holds. Then there are $\mathsf{MSO}[o]$-formulas ϕ_1 and ϕ_2 defining $\mathrm{wrt}^{-1}(R_1)$ and $\mathrm{rd}^{-1}(R_2)$, respectively. Set

$$\phi := \exists x_1, x_2\colon \phi_1\restriction_{x \leq_+ x_1} \wedge \phi_2\restriction_{x_2 \leq_- x} \wedge \neg P_{|w|}(x_1)\,.$$

Then we have $T = T(\phi)$. ◂

6.4.4 From Logical Definability To Recognizability

Finally, we have to prove that each $\mathsf{MSO}[o]$-definable plq language is recognizable. Concretely, we do this by translation of a formula $\phi \in \mathsf{MSO}[o]$ to a formula $\psi \in \mathsf{MSO}[\lambda]$ such that $\eta^{-1}(T(\phi)) = L(\psi)$ holds. In this case, the right-hand side of this equation is regular by [Büc60] implying that $T(\phi)$ is recognizable in $\mathbb{T}(\mathcal{Q}_\mathcal{L})$ since η is an epimorphism.

The most complicated case in our construction is the translation of the atomic formula $P_\ell(x)$ since write and read actions are commutative in certain contexts (cf. Lemma 4.7.2). For this translation we will utilize the connection between P_ℓ^w and $\mathrm{obw}(w)$ as described in Observation 6.4.26.

So, let $w \in \Sigma^*$ be a word, $p \in \mathrm{dom}(w)$ be a position in w (which will be represented by x in our formula), and $\ell \in \mathbb{N} \setminus \{0\}$. Then we express $p \in P_\ell^w$ as follows:

By definition we have $P_\ell^w \subseteq \Lambda_A^w$. Consequently, we have $(\underline{w}, p) \vDash \Lambda_A(x)$. Additionally, from Observation 6.4.26 we can infer two cases: if $|\mathrm{rd}(w)| < \ell$ then we are done. This property can be expressed with the help of an appropriate $\mathsf{FO}[\lambda]$-formula short_ℓ satisfying $L(\mathsf{short}_\ell) = \eta^{-1}(\mathrm{rd}^{-1}(A^{<\ell}))$. So, we can assume $|\mathrm{rd}(w)| \geq \ell$ from now on. Then, by Observation 6.4.26 we have $\mathrm{obw}(w) \leq \ell$, i.e., $[\![w]\!] \in \mathbb{T}(Q_{\mathcal{L}}) \setminus \Omega_\ell$. By Lemma 6.4.14 the plq language Ω_ℓ is aperiodic implying the existence of an $\mathsf{FO}[\lambda]$-formula Omega_ℓ satisfying $\eta^{-1}(\Omega_\ell) = L(\mathsf{Omega}_\ell)$.

Next, we want to determine the values $k := \mathrm{obw}(w)$ and $m := |\mathrm{rd}_2(w)|$ as well as the positions of the last k respectively m read actions and their corresponding write actions in w. To this end, we utilize Lemma 6.4.13 with some small modifications to the formulas embed_i and $\mathsf{overlap}_i$:

$$\mathsf{embed}'_i(\vec{x}, \vec{y}) := \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5 \qquad \text{and}$$
$$\mathsf{overlap}'_i(\vec{x}, \vec{y}) := \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5 \wedge \phi_6 \,.$$

Then we have $(\underline{w}, \vec{p}, \vec{q}) \vDash \mathsf{embed}'_i(\vec{x}, \vec{y})$ if, and only if, there is $r_2 \in A^i$ with $\mathrm{rd}(w) \in A^* r_2$, $w_1 \in \mathrm{redsup}_{\mathcal{L}}(r_2)$ for a prefix w_1 of $\mathrm{wrt}(w)$, where $p_1, \ldots, p_i$ are the positions of the last i read actions in w (in descending order), and $q_1, \ldots, q_i$ are the positions of their corresponding write actions in w. Moreover, we have $(\underline{w}, \vec{p}, \vec{q}) \vDash \mathsf{overlap}'_i(\vec{x}, \vec{y})$ if, and only if, in addition to the conditions above we have $\mathrm{rd}_2(w) \in A^* r_2$. Hence, we can express "$k = \mathrm{obw}(w)$" as follows:

$$\mathsf{obw}_k(\vec{x}, \vec{y}) := \mathsf{embed}'_k(\vec{x}, \vec{y}) \wedge \neg \mathsf{overlap}'_k(\vec{x}, \vec{y}) \wedge \bigwedge_{i=1}^{k-1} \mathsf{embed}_i \to \mathsf{overlap}_i \,.$$

Additionally, the property "$m = |\mathrm{rd}_2(w)| < k$" can be expressed by the following $\mathsf{FO}[\lambda]$-formula:

$$\mathsf{shuffle}_{m,k}(\vec{x}, \vec{z}) := \mathsf{overlap}'_m(\vec{x}, \vec{z}) \wedge \bigwedge_{i=m+1}^{k} \neg \mathsf{overlap}_i \,.$$

We should note here, that the property "$m = |\mathrm{rd}_2(w)|$" cannot be expressed by an $\mathsf{MSO}[\lambda]$-formula since this property is not recognizable according to [HKZ17, Observation 9.1] (at least in the case $F = \varnothing$). In contrast, satisfaction of $\mathsf{shuffle}_{m,k}(\vec{x}, \vec{z})$ by w requires further assumptions on the overlap's bounded width of w: we require that $k = \mathrm{obw}(w) < \infty$ is a fixed upper bound of $|\mathrm{rd}_2(w)|$.

Now, let $\vec{p}, \vec{q}, \vec{s} \in \mathrm{dom}(w)^\ell$ be positions in w with $(\underline{w}, \vec{p}, \vec{q}) \vDash \mathsf{shuffle}_{m,k}(\vec{x}, \vec{y})$ and $(\underline{w}, \vec{p}, \vec{s}) \vDash \mathsf{obw}_k(\vec{x}, \vec{z})$. It is easy to see that $q_i \leq s_i$ holds for each $1 \leq i \leq m$. In particular, if $q_i = s_i$ holds then we have $q_j = s_j$ for each $1 \leq j < i$. So, let $1 \leq i \leq m$ be minimal such that $q_{i+1} < s_{i+1}$ holds (where $q_{m+1} := 0$). This value can be determined with the help of the following formula:

$$\mathsf{diff}_i(\vec{y}, \vec{z}) \vDash \bigwedge_{j=1}^{i} y_j = z_j \wedge y_{i+1} < z_{i+1} \,,$$

where "$y_{m+1} < z_{m+1}$" means "true". Then by utilization of the equations from Lemma 4.7.2 we can move the read actions on positions $p_{i+1}, \ldots, p_k$ in w to the direct left-hand side

of the write action on position s_{i+1}. But it is impossible to transpose the read action on position p_{i+1} with its corresponding write action on position s_{i+1}. In other words, we have $P_\ell^w = P_k^w = \{p' \in \Lambda_A^w \mid s_{i+1} \leq p'\}$.

All in all, we set:

$$\mathsf{P}_\ell(x) := (\Lambda_A(x) \wedge \mathsf{short}_\ell) \vee \Big(\Lambda_A(x) \wedge \neg\mathsf{Omega}_\ell \wedge \forall \vec{x}, \vec{y}, \vec{z}\colon \bigwedge_{0 \leq i \leq m < k \leq \ell} \big(\mathsf{shuffle}_{m,k}(\vec{x}, \vec{y}) \wedge \mathsf{obw}_k(\vec{x}, \vec{z}) \wedge \mathsf{diff}_i(\vec{y}, \vec{z})\big) \to z_{i+1} \leq x\Big).$$

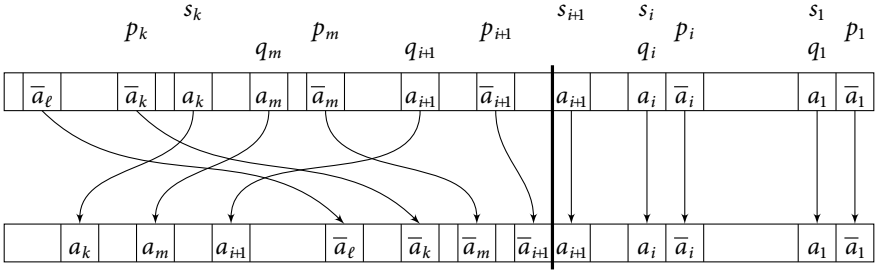

Figure 6.1. Visualization of a possible movement of read and write actions. In both words, we have $q_j < p_j$ for all $1 \leq j \leq m$ and $s_{i+1} \geq p_{i+1}$. Hence, both words have an overlap of length m. The write actions on the left-hand side of the bold line are contained in P_ℓ^w, the ones on the right-hand side are not contained in P_ℓ^w.

The next two lemmas prove the correctness and completeness of this formula $\mathsf{P}_\ell(x)$:

Lemma 6.4.29. *Let $\ell \in \mathbb{N} \setminus \{0\}$, $w \in \Sigma^*$, and $p \in P_\ell^w$. Then we have $(\underline{w}, p) \vDash \mathsf{P}_\ell(x)$.*

Proof. We prove the contraposition of this statement. So, let $p \in \mathrm{dom}(w)$ with $(\underline{w}, p) \nvDash \mathsf{P}_\ell(x)$. We show that $p \notin P_\ell^w$ holds in this case.

If we have $p \notin \Lambda_A^w$ we are done since $P_\ell^w \subseteq \Lambda_A^w$. So we can assume $p \in \Lambda_A^w$ from now on. Then, we have $\underline{w} \nvDash \mathsf{short}_\ell$ implying $|\mathrm{rd}(w)| \geq \ell$.

If $\underline{w} \vDash \mathsf{Omega}_\ell$ we have $\mathrm{obw}(w) > \ell$ and, hence, $P_\ell^w = \varnothing$ by Observation 6.4.26. In this case, we are done. Now, we assume $\underline{w} \vDash \neg\mathsf{Omega}_\ell$, i.e., $\mathrm{obw}(w) \leq \ell$. Then there are $\vec{p}, \vec{q}, \vec{s} \in \mathrm{dom}(w)^\ell$ and $0 \leq i \leq m < k \leq \ell$ such that

- $(\underline{w}, \vec{p}, \vec{q}, \vec{s}) \vDash \mathsf{shuffle}_{m,k}(\vec{x}, \vec{y}) \wedge \mathsf{obw}_k(\vec{x}, \vec{z}) \wedge \mathsf{diff}_i(\vec{y}, \vec{z})$ and
- $(\underline{w}, p, \vec{s}) \nvDash z_{i+1} \leq x$.

As we have argued above, we have $m = |\mathrm{rd}_2(w)|$ and $k = \mathrm{obw}(w)$. Additionally, we have:

- $p_1, \ldots, p_k$ are the positions of the last k read actions in w (in descending order),
- $q_1, \ldots, q_m$ are the write positions corresponding to the read actions on $p_1, \ldots, p_m$, and
- $s_1, \ldots, s_k$ are the write positions corresponding to the read actions on $p_1, \ldots, p_k$.

By definition of diff_i the number i is the minimal index such that $q_{i+1} < s_{i+1}$ holds (where $q_{m+1} := 0$). We have to prove that $p < s_{i+1}$ implies $p \notin P_\ell^w$. To this end, let $r_1, r_2, w_1, w_2 \in A^*$ such that $\mathrm{rd}(w) = r_1 r_2$, $|r_2| = i$, $\mathrm{wrt}(w) = w_1 w_2$, and $w_1 = \mathrm{wrt}(w[1, s_{i+1} - 1])$. We first show $w \equiv w_1 \overline{r_1} w_2 \overline{r_2}$. By Theorem 4.7.8 it suffices to prove $|\mathrm{rd}_2(w)| = |\mathrm{rd}_2(w_1 \overline{r_1} w_2 \overline{r_2})|$. We have

- the last i read actions are on the right-hand side of each write action and, hence, on the right-hand side of their corresponding write actions and
- the read actions on positions $p_{i+1}, \ldots, p_k$ are on the right-hand side of their corresponding write actions since $q_{i+1} < s_{i+1}$, i.e., the write actions on positions $q_{i+1}, \ldots, q_k$ in w are contained in w_1 which is on the left-hand side of all read actions.

Hence, we have $\underline{w_1 \overline{r_1} w_2 \overline{r_2}} \vDash \mathsf{overlap}_m$ implying $|\mathrm{rd}_2(w)| = m \leq |\mathrm{rd}_2(w_1 \overline{r_1} w_2 \overline{r_2})|$. However, we have $\underline{w_1 \overline{r_1} w_2 \overline{r_2}} \nvDash \mathsf{overlap}_k$ since the read action on position p_{i+1} in w (which is the last letter in $\overline{r_1}$) is on the left-hand side of its corresponding write action on position s_{i+1} (which is the first letter in w_2). Hence, we have $|\mathrm{rd}_2(w_1 \overline{r_1} w_2 \overline{r_2})| < k = \mathrm{obw}(w)$. By minimality of $\mathrm{obw}(w)$ we can infer that $|\mathrm{rd}_2(w)| \geq |\mathrm{rd}_2(w_1 \overline{r_1} w_2 \overline{r_2})|$ holds. Therefore, we have $w \equiv w_1 \overline{r_1} w_2 \overline{r_2}$.

By $p < s_{i+1}$ we can split $w_1 \overline{r_1} w_2 \overline{r_2}$ as follows: let $v_1, v_2 \in A^*$ with $v_1 = \mathrm{wrt}(w[1, p])$, i.e., v_1 is a prefix of w_1, and v_2 is the complementary suffix of $w_1 \overline{r_1} w_2 \overline{r_2}$ wrt. v_1. Then we have $w \equiv w_1 \overline{r_1} w_2 \overline{r_2} = v_1 v_2$ and $|\mathrm{rd}(v_2)| = |\mathrm{rd}(w)| \geq \ell$ implying $p \notin P_\ell^w$. ◂

Lemma 6.4.30. *Let $\ell \in \mathbb{N} \setminus \{0\}$, $w \in \Sigma^*$, and $p \in \mathrm{dom}(w) \setminus P_\ell^w$. Then we have $(\underline{w}, p) \nvDash \mathsf{P}_\ell(x)$.*

Proof. If we have $p \notin \Lambda_A^w$ we have obviously $(\underline{w}, p) \nvDash \mathsf{P}_\ell(x)$. So, from now on, we assume $p \in \Lambda_A^w$.

By $p \in \mathrm{dom}(w) \setminus P_\ell^w$ there are $v_1, v_2 \in \Sigma^*$ with $w \equiv v_1 v_2$, $\mathrm{wrt}(v_1) = \mathrm{wrt}(w[1, p])$, and $|\mathrm{rd}(v_2)| \geq \ell$. Hence, we have $|\mathrm{rd}(w)| \geq |\mathrm{rd}(v_2)| \geq \ell$ implying $\underline{w} \nvDash \mathsf{short}_\ell$.

Next, we consider the value of $\mathrm{obw}(w)$. If $\mathrm{obw}(w) > \ell$, we have $[\![w]\!] \in \Omega_\ell$ implying $\underline{w} \vDash \mathsf{Omega}_\ell$. Hence, we are done in this case. Now, assume $\mathrm{obw}(w) \leq \ell$. Then there are $0 \leq m < k \leq \ell$ with $\mathrm{obw}(w) = k$ and $|\mathrm{rd}_2(w)| = m$. Let $\vec{p}, \vec{q}, \vec{s} \in \mathrm{dom}(w)^\ell$ be the following positions:

- $p_1, \ldots, p_k$ are the positions of the last k read actions in w (in descending order),
- $q_1, \ldots, q_m$ are the write positions corresponding to the read actions on $p_1, \ldots, p_m$, and
- $s_1, \ldots, s_k$ are the write positions corresponding to the read actions on $p_1, \ldots, p_k$.

Since we have $q_i \leq s_i$ for each $1 \leq i \leq m$, there is an index $0 \leq i \leq m$ such that $q_{i+1} < s_{i+1}$ holds (where $q_{m+1} := 0$). Let $0 \leq i \leq m$ be minimal with $q_{i+1} < s_{i+1}$. Then we have

$$(\underline{w}, \vec{p}, \vec{q}, \vec{s}) \vDash \mathsf{shuffle}_{m,k}(\vec{x}, \vec{y}) \wedge \mathsf{obw}_k(\vec{x}, \vec{z}) \wedge \mathsf{diff}_i(\vec{y}, \vec{z}) \,.$$

Now, we prove that $(\underline{w}, p, \vec{s}) \nvDash z_{i+1} \leq x$, i.e., we prove $s_{i+1} > p$. Towards a contradiction, suppose that $s_{i+1} \leq p$ holds.

Recall that $v_1, v_2 \in \Sigma^*$ are defined such that $w \equiv v_1 v_2$, $\mathrm{wrt}(v_1) = \mathrm{wrt}(w[1, p])$, and $|\mathrm{rd}(v_2)| \geq \ell$. Consider the positions in $v_1 v_2$ of the letters on positions p, $\vec{p}$, $\vec{q}$, and $\vec{s}$ in w.

To this end, let σ be the permutation of $\mathrm{dom}(w)$ mapping the positions in $\underline{w}$ to the ones in $\underline{v_1}\underline{v_2}$. Then we have $\sigma(q_j) < \sigma(p_j)$ for each $1 \leq j \leq m$ since $\mathrm{rd}_2(w) = \mathrm{rd}_2(v_1v_2)$ and, therefore, $\underline{v_1}\underline{v_2} \vDash \mathsf{overlap}_m$ holds.

By $\mathrm{wrt}(v_1) = \mathrm{wrt}(w[1,p])$ we have $\sigma(p) \leq |v_1|$ and, by $|\mathrm{rd}(v_2)| \geq \ell \geq k$ we have $|v_1| < \sigma(p_j)$ for each $1 \leq j \leq k$. From $w \equiv v_1v_2$ we know that $\mathrm{wrt}(w) = \mathrm{wrt}(v_1v_2)$ and $\mathrm{rd}(w) = \mathrm{rd}(v_1v_2)$ hold by Theorem 4.7.8 implying

$$\begin{aligned}
&\sigma(s_k) < \cdots < \sigma(s_{i+1}) && \text{(since } s_k <_+ s_{k-1} <_+ \cdots <_+ s_{i+1}\text{)}\\
&\leq \sigma(p) && \text{(since } s_{i+1} \leq p,\ s_{i+1}, p \in \Lambda_A^w\text{, and, therefore, } s_{i+1} \leq_+ p\text{)}\\
&\leq |v_1| && \text{(since } \mathrm{wrt}(v_1) = \mathrm{wrt}(w[1,p])\text{)}\\
&< \sigma(p_k) < \cdots < \sigma(p_1)\,. && \text{(since } |\mathrm{rd}(v_2)| \geq k \text{ and } p_k <_- p_{k-1} <_- \cdots <_- p_1\text{)}
\end{aligned}$$

Recall that we also have $\sigma(s_j) = \sigma(q_j) < \sigma(p_j)$ for each $1 \leq j \leq i$ by the minimality of $i \leq m$ and due to $\underline{v_1}\underline{v_2} \vDash \mathsf{overlap}_m$. Then we also obtain $\sigma(s_j) < \sigma(p_j)$ for each $1 \leq j \leq k$, i.e., $\underline{v_1}\underline{v_2} \vDash \mathsf{overlap}_k$. From Lemma 6.4.13 we infer that $|\mathrm{rd}_2(v_1v_2)| \geq k > m = |\mathrm{rd}_2(w)|$ holds. This is a contradiction to $\mathrm{rd}_2(v_1v_2) = \mathrm{rd}_2(w)$ by $w \equiv v_1v_2$ according to Theorem 4.7.8. Hence, we have $p < s_{i+1}$ implying $(\underline{w}, p) \nvDash \mathsf{P}_\ell(x)$. ◂

Finally, with the help of the formula P_ℓ we can prove the last implication of Theorem 6.4.1:

Proposition 6.4.31. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet with $|F| + |U| \geq 2$ and $T \subseteq \mathbb{T}(Q_\mathcal{L})$ be $\mathsf{MSO}[o]$-definable. Then T is recognizable.*

Proof. Let $T \subseteq \mathbb{T}(Q_\mathcal{L})$ be $\mathsf{MSO}[o]$-definable. Then there is $\phi \in \mathsf{MSO}[o]$ with $T = T(\phi)$. We construct a formula $\phi' \in \mathsf{MSO}[\lambda]$ such that for each $w \in \Sigma^*$ we have $\widetilde{w} \vDash \phi$ if, and only if, $\underline{w} \vDash \phi'$.

Concretely, we obtain ϕ' from ϕ as follows:

(1) replace any occurrence of $x <_+ y$ by $x < y \wedge \Lambda_A(x) \wedge \Lambda_A(y)$,

(2) replace any occurrence of $x <_- y$ by $x < y \wedge \Lambda_{\overline{A}}(x) \wedge \Lambda_{\overline{A}}(y)$, and

(3) replace any occurrence of $P_\ell(x)$ by the $\mathsf{FO}[\lambda]$-formula $\mathsf{P}_\ell(x)$.

Then by Lemmata 6.4.29 and 6.4.30 we have $\widetilde{w} \vDash \phi$ if, and only if, $\underline{w} \vDash \phi'$ for any $w \in \Sigma^*$. Hence, by Büchi's Theorem [Büc60] $\eta^{-1}(T)$ is regular. Since η is surjective this implies recognizability of $T = T(\phi)$. ◂

6.4.5 The Complexity of the Constructions in Theorem 6.4.1

Finally, we want to analyze the complexities of the constructions in this section. Recall that we have seen the following circular chain of implications: "(A)/(i) ⇒ (ii) ⇒ (iii) ⇒ (B) ⇒ (C) ⇒ (A)".

Towards the first implication, let $T \subseteq \mathbb{T}(Q_\mathcal{L})$ be recognized by a finite monoid $\mathbb{F}$ via the homomorphism $\phi\colon \mathbb{T}(Q_\mathcal{L}) \to \mathbb{F}$. Then $\eta^{-1}(T) \subseteq \Sigma^*$ is regular and recognized by $\mathbb{F}$ via $\eta \circ \phi$ (recall that we can understand $\mathbb{F}$ as a DFA accepting $\eta^{-1}(T)$). We can obtain another finite monoid $\mathbb{G}$ recognizing $\eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ in polynomial time with the help of the

cross-product construction known from automata theory. Note that $|\mathbb{G}| \in O(|\mathbb{F}|)$ holds in this case.

Now, consider the second implication. So, let $\eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ be recognized by a finite monoid $\mathbb{F}$. By the proof of Lemma 6.4.10, the set T is a union of $O(2^{|\mathbb{F}|})$ many languages which can be expressed by a constant number of plq languages of the form $\mathrm{wrt}^{-1}(R)$, $\mathrm{rd}^{-1}(R)$, and Ω_ℓ for regular languages $R \subseteq A^*$ and numbers $\ell \in \mathbb{N}$. In other words, the constructed Boolean combination has exponential size.

Next, let $T \subseteq \mathbb{T}(\mathcal{Q}_{\mathcal{L}})$ be given by a Boolean combination as in property (iii) of Theorem 6.4.6. If a set $\mathrm{wrt}^{-1}(R)$ or $\mathrm{rd}^{-1}(R)$ is given by a finite automaton accepting $R \subseteq A^*$ then a q-rational expression describing such language has exponential size [EZ74]. Otherwise if such set is given by a regular expression, an equivalent q-rational expression has linear size. Additionally, the plq language Ω_ℓ is equivalent to a q-rational expression of size exponential in the number ℓ (cf. Proposition 6.4.22).

Towards the implication "(B)⇒(C)", let $T \subseteq \mathbb{T}(\mathcal{Q}_{\mathcal{L}})$ be given by a q-rational expression. The translation of q^+- and q^--rational expressions into an $\mathsf{MSO}[o]$-formula is similar to the translation of a regular expression into an $\mathsf{MSO}[\lambda]$-formula. This is possible in polynomial time when translating the given regular expression into an NFA (this uses the efficient closure properties of regular languages) and then into an $\mathsf{MSO}[\lambda]$-formula as described (e.g.) in [Lib04, Theorem 7.21]. Finally, formulas for unions, complements, and products can be constructed in polynomial time. All in all, we obtain an $\mathsf{MSO}[o]$-formula from a q-rational expression in polynomial time.

Finally, let $T \subseteq \mathbb{T}(\mathcal{Q}_{\mathcal{L}})$ be given by an $\mathsf{MSO}[o]$-formula. In Proposition 6.4.31 we translated this formula into an $\mathsf{MSO}[\lambda]$-formula. The most complicated case is the atomic formula $P_\ell(x)$. Its translation $\mathsf{P}_\ell(x)$ can be obtained from $P_\ell(x)$ in time exponential in the number ℓ. The remaining translations are possible in linear time. Finally, the translation of an $\mathsf{MSO}[\lambda]$-formula into a DFA is not elementary [SM73].

6.5 Conclusion

We have characterized the recognizable languages of the plq and pls monoid. First, we have proved that most pls monoids have only the trivial recognizable languages: the empty set and the monoid itself. Only the partially lossy stack monoids with a unary underlying lossiness alphabet have an infinite number of recognizable languages. These sets are recognized by finite cyclic groups.

We have also seen that - in contrast to the free monoid [Kle51] - the classes of rational and recognizable languages in the plq monoid do not coincide. This holds since the class of recognizable plq languages is not closed under product and iteration. But we have introduced a modification on the rational expressions - so-called *q-rational* expressions: starting from the sets having read or write action sequences from a regular language, we build the closure under Boolean operations and restricted versions of the product and iteration. We have seen that the class of recognizable languages in the partially lossy queue monoid is exactly the class of q-rational languages. However, it is still unknown whether there also is such kind of expressions with monotonic operations, only (i.e., without the usage of complement and intersection).

Moreover, we have defined a Büchi-like monadic second-order logic on the positions of actions (cf. [Büc60]). This logic describes exactly the recognizable plq languages. Unfortu-

nately, the signature of this logic is infinite, but elements of the plq monoid are at least finitely presentable. Until now we do not know whether there also is a monadic second-order logic with just a finite number of relations.

Chapter 7

Aperiodic and Star-Free Languages

7.1 Introduction

Probably the most famous proper subclass of the regular languages is the class of aperiodic languages (see Definition 5.2.3). This class shares a bunch of properties with the class of regular languages. So it is closed under Boolean operations, concatenation, and homomorphic preimages. Though, it is not closed under iteration. One of the most important results in automata theory is Schützenberger's Theorem [Sch65] stating that a word language is aperiodic if, and only if, it is star-free. However, there are even more characterizations of the aperiodic word languages. The aperiodic languages are the languages: (1) which have an aperiodic syntactic monoid, (2) which are accepted by a so-called *counter-free* finite automaton (these are NFAs which are unable to count modulo any positive integer) [MP71], (3) which are described by the first-order fragment of Büchi's logic on words [MP71], and (4) which are definable in linear temporal logic [Kam68]. From [GRS91] we also know that the classes of star-free and aperiodic trace languages coincide. Hence, it is worth to study the class of aperiodic languages in the transformation monoid of a partially lossy queue or stack.

In the previous chapter we have learned that the class of recognizable languages in the plq monoid is not closed under the monoid's product. Since this non-closure property also holds for the aperiodic plq languages, this class also is not closed under product. Hence, the classes of aperiodic and star-free languages do not coincide in the partially lossy queue monoid. But we will see that we can generate the aperiodic plq languages by star-free expressions with the same restriction to the monoid's product as in our previous result regarding rational expressions. Additionally, we learn that the class of aperiodic plq languages can be described by first-order formulas of our special queue logic.

But before, we will see that the aperiodic subsets of the partially lossy stack monoid are only the trivial ones and this time this statement holds in any case.

7.2 Partially Lossy Stacks

Recall that the partially lossy stack monoid with an at least binary underlying lossiness alphabet has exactly two recognizable pls languages: the monoid itself and the empty set. Since any aperiodic language also is recognizable (recall that this is an explicit requirement in the definition), it is no surprise that in this case there also are only these two trivial aperiodic pls languages.

Now, consider a unary lossiness alphabet. Then from Theorem 6.2.2 we know that a pls language is in this case recognizable if, and only if, it is recognized by a finite cyclic group. Hence, there is an infinite number of recognizable pls languages. However, with recognizable

pls languages we are able to count modulo a positive integer in a finite, cyclic group. Since counting is not possible in aperiodic languages, we will see that almost all recognizable pls languages are not aperiodic - only the two trivial pls languages are aperiodic. This is the following theorem:

Theorem 7.2.1. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $T \subseteq \mathbb{T}(\mathcal{P}_\mathcal{L})$. Then T is aperiodic if, and only if, $T = \varnothing$ or $T = \mathbb{T}(\mathcal{P}_\mathcal{L})$.*

Proof. This claim is trivial if $A = F \cup U$ is an at least binary alphabet by Theorem 6.2.4. So, from now on, we consider the case $|A| = 1$.

Obviously the languages $\varnothing$ and $\mathbb{T}(\mathcal{P}_\mathcal{L})$ are aperiodic. Now, let $\varnothing \subsetneq T \subsetneq \mathbb{T}(\mathcal{P}_\mathcal{L})$ be aperiodic. Since T also is recognizable its syntactic monoid $\mathbb{S}(T)$ is a finite, cyclic group, i.e., $\mathbb{S}(T) \cong \mathbb{Z}_k$ for a positive integer $k > 0$ due to Theorem 6.2.2. Additionally, since T is aperiodic, $\mathbb{S}(T) \cong \mathbb{Z}_k$ is an aperiodic monoid, i.e., there is a number $n \in \mathbb{N}$ such that for each $\ell \in \mathbb{Z}_k$ we have $\ell^n = \ell^{n+1}$. This is only possible for $k = 1$ and $\ell = 0$. Hence, $|\mathbb{S}(T)| = 1$ holds implying $T = \varnothing$ or $T = \mathbb{T}(\mathcal{P}_\mathcal{L})$. ◂

We have seen that in any case the pls monoid has exactly two aperiodic languages. We may also ask for a characterization of the star-free pls languages. So, we conjecture that this class of pls languages can be characterized similar to the rational languages (cf. Theorem 5.3.1). In other words, possibly a language $T \subseteq \mathbb{T}(\mathcal{P}_\mathcal{L})$ is star-free if, and only if there is a star-free word language $L \subseteq \mathrm{NF}_{\mathcal{P}_\mathcal{L}} = \overline{A}^* A^* \cup \{\perp\!\!\!\perp\}$ with $T = \eta(L)$.

7.3 Partially Lossy Queues

Now, we consider the aperiodic and star-free languages in the partially lossy queue monoid. To this end, we recall the proof of Proposition 6.4.16. We have seen in that proof that $[\![a]\!]^* \cdot [\![\overline{a}]\!]^*$ is no recognizable plq language. Since $[\![a]\!]^*$ and $[\![\overline{a}]\!]^*$ are even aperiodic, we learn that the class of aperiodic plq languages is not closed under product and, hence, does not coincide with the class of star-free plq languages. To describe the aperiodic plq languages in terms of star-free expressions, we have to adopt our special restriction to the monoid's product from rational expressions. With this restriction we obtain the so-called *q-star-free* expressions which are formally defined as follows:

Let $\mathcal{L} = (F, U)$ be a lossiness alphabet. A $\mathbb{T}(\mathcal{Q}_\mathcal{L})$-language is *$q^+$-star-free* if it can be constructed by the following rules:

(1^+) $\mathrm{wrt}^{-1}(\varepsilon)$, $\mathrm{wrt}^{-1}(\varnothing) = \varnothing$, and $\mathrm{wrt}^{-1}(a)$ for any $a \in A$ are q^+-star-free and

(2^+) if $S, T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ are q^+-star-free then $S \cup T$, $S \cdot T$, and $\mathbb{T}(\mathcal{Q}_\mathcal{L}) \setminus S$ are q^+-star-free.

Similarly, by replacing wrt^{-1} by rd^{-1} in the rules above, we define the class of *q^--star-free* $\mathbb{T}(\mathcal{Q}_\mathcal{L})$-languages. Finally, a language in $\mathbb{T}(\mathcal{Q}_\mathcal{L})$ is *q-star-free* if it can be constructed from the following rules:

(1) if $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ is q^+- or q^--star-free it also is q-star-free,

(2) if $S, T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ are q-star-free then $S \cup T$ and $\mathbb{T}(\mathcal{Q}_\mathcal{L}) \setminus S$ are q-star-free, and

(3) if $S \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ is q^+-star-free and $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ is q^--star-free such that $\mathrm{rd}(T)$ is finite (i.e, T is obtained without usage of the $\setminus$-operator) then $S \cdot \mathbb{T}(\mathcal{Q}_\mathcal{L}) \cdot T$ is q-star-free.

Hence, the only difference of the rules above in comparison to the ones constructing the q-rational plq languages is the replacement of the iteration by complement in the definition of q^+-star-free languages. Therefore, $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ is q^+-star-free (q^--star-free) if, and only if, there is an aperiodic language $R \subseteq A^*$ with $T = \mathrm{wrt}^{-1}(R)$ (respectively $T = \mathrm{rd}^{-1}(R)$).

Similarly to Theorems 6.4.1 and 6.4.6, we can state the following result:

Theorem 7.3.1. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$. Then the following statements are equivalent:*

(A) T is aperiodic.

(B) $\eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ is aperiodic.

(C) T is a Boolean combination of plq languages of the form $\mathrm{wrt}^{-1}(R)$, $\mathrm{rd}^{-1}(R)$, and Ω_ℓ for aperiodic languages $R \subseteq A^*$ and numbers $\ell \in \mathbb{N}$.

(D) T is q-star-free.

(E) T is $\mathsf{FO}[o]$-definable.

Proof. To prove this theorem we recall the proofs of Theorems 6.4.1 and 6.4.6.

At first we show the implication "(A)⇒(B)". So, let $T \subseteq \mathbb{T}(\mathcal{Q}_\mathcal{L})$ be aperiodic. Then $\eta^{-1}(T)$ is aperiodic. Since $\overline{A}^* A^* \overline{A}^*$ is aperiodic and the class of aperiodic languages is closed under intersection we can infer that $\eta^{-1}(T) \cap \overline{A}^* A^* \overline{A}^*$ also is aperiodic.

Next, we prove "(B)⇒(C)". We recall the proof of Lemma 6.4.10 and assume that the recognizing monoid $\mathbb{F}$ is aperiodic. All of the arguments of the inverse projections wrt^{-1} and rd^{-1} are either single words, $\mu^{-1}(x)$, $\overline{\mu}^{-1}(x)$ for any $x \in \mathbb{F}$, or products of those languages. Anyway these plq languages are aperiodic since aperiodicity is preserved under inverse homomorphisms (note that the class of aperiodic word languages is closed under concatenation according to Schützenberger's Theorem [Sch65]). Hence, T is a Boolean combination of plq languages of the form $\mathrm{wrt}^{-1}(R)$, $\mathrm{rd}^{-1}(R)$, and Ω_ℓ for aperiodic languages $R \subseteq A^*$ and numbers $\ell \in \mathbb{N}$.

In the proof of Proposition 6.4.22 we see that Ω_ℓ is even q-star-free and, hence, the implication "(C)⇒(D)" holds.

In the construction of the proof of Proposition 6.4.28 each of the used formulas can be expressed in first-order logic. Therefore, we have "(D)⇒(E)".

Finally, we have to prove "(E)⇒(A)". Each translation of an $\mathsf{FO}[o]$-formula as seen in Proposition 6.4.31 results in a formula in $\mathsf{FO}[\lambda]$. Therefore, by [MP71] each $\mathsf{FO}[o]$-definable plq language is aperiodic. ◂

There are still several open questions concerning aperiodic and star-free plq languages. For example, one may ask for a temporal logic describing the aperiodic languages similar to Kamp's Theorem [Kam68] stating that the logic LTL describes exactly the aperiodic word

languages. One could also study subclasses of the star-free and aperiodic plq languages. For example, one may consider the dot-depth hierarchy or the Straubing-Thérien hierarchy. Thomas proved in [Tho82] that in the free monoid the n^{th} level of the former hierarchy corresponds to the $\mathcal{B}\Sigma_n$-fragment of Büchi's logic on words and that the $(n - \frac{1}{2})^{\text{th}}$ level corresponds to the Σ_n-fragment. It is an open question whether there exists also a modification on this hierarchy such that the named correspondence holds for our queue logic $\mathsf{FO}[o]$.

Moreover, it is open whether aperiodicity or star-freeness of a given rational, recognizable, or star-free language in $\mathbb{T}(\mathcal{Q}_\mathcal{I})$ is decidable. One may also check whether the undecidable problems from Section 5.4 still are undecidable if their inputs are star-free plq languages.

7.4 Conclusion

All in all, we have characterized the aperiodic languages in the pls and plq monoids. Concretely, we have seen that the partially lossy stack monoid has no non-trivial aperiodic languages. Towards the aperiodic plq languages we revisited the proofs of Theorems 6.4.1 and 6.4.6 and found some small modifications such that these theorems also hold for aperiodic plq languages.

Part II
Reachability Problems and Verification

Chapter 8

Reachability Problems and Temporal Logics

In this chapter we recall some basic definitions and knowledge about verification of finite- and infinite-state systems. We will start with simplifying our automata with storage mechanisms. Concretely, we will remove their input tape and their initial and final states. Afterwards we introduce multiple reachability problems and some temporal logics like LTL and CTL on these systems. Finally, we give an idea of the decidability proofs of the model checking problem for various temporal logics. In the succeeding chapters we will finally show various decidability results on the named problems.

8.1 Kripke-Structures

First, we introduce so-called Kripke-structures. These are essentially directed graphs with labels on their nodes. We use them later to define the semantics of the temporal logics.

Definition 8.1.1. Let AP be a finite set of atomic propositions. A *Kripke-structure* is a triple $\mathcal{M} = (V, E, \Lambda)$ where V is a (not necessarily finite) set of *nodes*, $E \subseteq V^2$ is a set of *edges*, and $\Lambda \colon V \to 2^{\mathrm{AP}}$ is a *labeling*. ⌟

Now, let $\mathcal{M} = (V, E, \Lambda)$ be a Kripke-structure. An *(infinite) path* in $\mathcal{M}$ is a sequence $(v_i)_{i \in \mathbb{N}}$ with $v_i \to_{\mathcal{M}} v_{i+1}$ for each $i \in \mathbb{N}$. For a path $\rho = (v_i)_{i \in \mathbb{N}}$ in $\mathcal{M}$ and $n \in \mathbb{N}$, we denote the subpath of ρ starting after n steps by $\rho^n := (v_i)_{i \geq n}$.

Next, we want to obtain Kripke-structures from finite automata having a memory modeled by the data type $\mathcal{D}$. To this end, we have to simplify these automata:

Definition 8.1.2. Let $\mathcal{D} = (U, \Sigma, \Theta)$ be a data type and AP be a set of atomic propositions. A *$\mathcal{D}$-system* is a tuple $\mathfrak{S} = (Q, \mathcal{D}, \Delta, \Lambda)$ where Q is a finite set of *states*, $\Delta \subseteq Q \times \Sigma^* \times Q$ is a finite set of *transitions*, and $\Lambda \colon Q \to 2^{\mathrm{AP}}$ is a *labeling*. ⌟

In other words, a $\mathcal{D}$-system is a $\mathcal{D}$-automaton without input tape, initial states, and final states. Instead of that, systems have an additional labeling of their control states. Later we will use such labeling to define the semantics of temporal logics.

From these systems we can obtain a Kripke-structure as follows:

Definition 8.1.3. Let $\mathcal{D} = (U, \Sigma, \Theta)$ be a data type and $\mathfrak{S} = (Q, \mathcal{D}, \Delta, \Lambda)$ be a $\mathcal{D}$-system. The system $\mathfrak{S}$ induces a (possibly infinite) Kripke-structure $\mathcal{M}(\mathfrak{S}) := (V, E, \Lambda')$ where

- $V = \mathrm{Conf}_{\mathfrak{S}} := Q \times U$,
- $((p, x), (q, y)) \in E$ if, and only if, there is $(p, t, q) \in \Delta$ with $[\![t]\!](x) = y$, and

- $\Lambda'((q,x)) := \Lambda(q)$.

Hence, the Kripke-structure $\mathcal{M}(\mathfrak{S})$ induced by a $\mathcal{D}$-system $\mathfrak{S}$ is essentially the configuration graph of $\mathfrak{S}$. In other words, from a given $\mathcal{D}$-automaton $\mathfrak{A}$ we obtain this Kripke-structure by removing the edge-labels from its configuration graph $\mathcal{G}_{\mathfrak{A}}$.

Similarly, we can define the Kripke-structure induced by a lossy $\mathcal{D}$-system (where $\mathcal{D}$ is a lossy data type). In this case, we have an edge $((p,x),(q,y)) \in E$ if, and only if, there is either a transition (p,t,q) with $y \in [\![t]\!](x)$ or $p = q$ and $y \leq x$ holds.

8.2 Reachability Problems

Now, we want to introduce several verification problems. Some of the most important questions in verification are listed below. We will see afterwards, that all of these problems are related to the reachability problem.

So, let $\mathcal{M} = (V, E, \Lambda)$ be a Kripke-structure.

(1) *Reachability Problem.* Given two sets $I, F \subseteq V$ of nodes. Can we reach F from I in $\mathcal{M}$?

(2) *Safety Problem.* Given two sets $I, F \subseteq V$ of nodes. Can we only reach F from I in $\mathcal{M}$?

(3) *Liveness Problem.* Given two sets $I, F \subseteq V$ of nodes. Can we reach F infinitely often on an infinite path in $\mathcal{M}$ starting in I?

(4) *Inevitability Problem.* Given two sets $I, F \subseteq V$ of nodes. Do all (infinite) paths in $\mathcal{M}$ starting in I stay in I until they eventually reach F once?

A possible approach to solve the aforementioned problems is to compute the set of forwards and backwards reachable nodes in our given Kripke-structure and to check afterwards, whether the respective properties hold. To this end, we first define these sets as follows:

Definition 8.2.1. Let $\mathcal{M} = (V, E, \Lambda)$ be a Kripke-structure and $U \subseteq V$ be a set of nodes in $\mathcal{M}$. We inductively define the functions $\mathrm{post}^i_{\mathcal{M}}(U)$ and $\mathrm{pre}^i_{\mathcal{M}}(U)$ as follows: $\mathrm{post}^0_{\mathcal{M}}(U) = \mathrm{pre}^0_{\mathcal{M}}(U) := U$. For $i \geq 0$ we set

$$\mathrm{post}^{i+1}_{\mathcal{M}}(U) := \{v \in V \mid \mathrm{post}^i_{\mathcal{M}}(U) \to_{\mathcal{M}} v\} \quad \text{and}$$
$$\mathrm{pre}^{i+1}_{\mathcal{M}}(U) := \{v \in V \mid v \to_{\mathcal{M}} \mathrm{pre}^i_{\mathcal{M}}(U)\}.$$

The set of *forwards and backwards reachable nodes* of $\mathcal{M}$ wrt. U are

$$\mathrm{post}^*_{\mathcal{M}}(U) := \bigcup_{i \in \mathbb{N}} \mathrm{post}^i_{\mathcal{M}}(U) \quad \text{and} \quad \mathrm{pre}^*_{\mathcal{M}} := \bigcup_{i \in \mathbb{N}} \mathrm{pre}^i_{\mathcal{M}}(U).$$

Let $\mathfrak{S}$ be a $\mathcal{D}$-system. To simplify notations we write $\mathrm{post}_{\mathfrak{S}}$ and $\mathrm{pre}_{\mathfrak{S}}$ instead of $\mathrm{post}_{\mathcal{M}(\mathfrak{S})}$ or $\mathrm{pre}_{\mathcal{M}(\mathfrak{S})}$.

By definition $\mathrm{post}^*_{\mathcal{M}}(U)$ and $\mathrm{pre}^*_{\mathcal{M}}(U)$ are infinite unions. However, we could also write these sets as follows:

$$\mathrm{post}^*_{\mathcal{M}}(U) = \{v \in V \mid U \to^*_{\mathcal{M}} v\} \quad \text{and} \quad \mathrm{pre}^*_{\mathcal{M}}(U) = \{v \in V \mid v \to^*_{\mathcal{M}} U\}.$$

Similarly, we may define $\mathrm{post}^+_{\mathcal{M}}(U)$ and $\mathrm{pre}^+_{\mathcal{M}}(U)$ which are the nodes in $\mathcal{M}$ which are forwards or backwards reachable in at least one step.

Some of the problems listed above can be directly solved by computation of the forwards or backwards reachable nodes in our Kripke-structure $\mathcal{M}$. Concretely, possible strategies are the following:

(1) To solve the reachability problem, we can compute the set $\mathrm{post}^*_{\mathcal{M}}(I)$ and decide whether $\mathrm{post}^*_{\mathcal{M}}(I) \cap F \neq \varnothing$ holds. Similarly, we could check, whether $\mathrm{pre}^*_{\mathcal{M}}(F) \cap I \neq \varnothing$ holds. In both cases, we require that the emptiness problem for intersections of sets is decidable.

(2) We can check safety by deciding whether $\mathrm{post}^*_{\mathcal{M}}(I) \subseteq F$ holds. Here, we need the decidability of the inclusion problem in V.

However, solving liveness and inevitability are a bit more involved. But we can express these properties in LTL which we will introduce in the next section.

Before we do this, we want to give an alternative definition of the forwards and backwards reachable configurations in a $\mathcal{D}$-system $\mathfrak{S}$. In this definition we remove the control states of $\mathfrak{S}$ and focus only on the contents of the attached data type $\mathcal{D}$. These are the following sets:

Definition 8.2.2. Let $\mathcal{D} = (U, \Sigma, \Theta)$ be a data type, $L \subseteq U$ be a set of contents of the data type, and $T \subseteq \Sigma^*$ be a language of action sequences. The set of *forwards reachable contents* of $\mathcal{D}$ wrt. L is

$$\textsc{Reach}_{\mathcal{D}}(L, T) = [\![T]\!](L) \setminus \{\bot\}$$

and the set of *backwards reachable contents* of $\mathcal{D}$ wrt. L is

$$\textsc{BackReach}_{\mathcal{D}}(L, T) = \{x \in U \mid [\![T]\!](x) \cap L \neq \varnothing\}\,.$$

Using the definition of $\mathcal{D}$'s semantics $[\![.]\!]$ we can show that Reach and BackReach are compatible with the composition of action sequences:

Observation 8.2.3. *Let $\mathcal{D} = (U, \Sigma, \Theta)$ be a data type, $L \subseteq U$, and $S, T \subseteq \Sigma^*$. Then we have the following statements:*

(1) $\textsc{Reach}(L, ST) = \textsc{Reach}(\textsc{Reach}(L, S), T)$.

(2) $\textsc{BackReach}(L, ST) = \textsc{BackReach}(\textsc{BackReach}(L, T), S)$.

Proof. We first prove (1). Let $x \in \textsc{Reach}(L, ST)$. Then there are $y \in L$, $s \in S$, and $t \in T$ with $\bot \neq x = [\![st]\!](y)$. Set $z := [\![s]\!](y) \neq \bot$. Then we have $z \in \textsc{Reach}(L, S)$ and $[\![t]\!](z) = [\![st]\!](y) = x$. This implies $x \in \textsc{Reach}(\{z\}, T) \subseteq \textsc{Reach}(\textsc{Reach}(L, S), T)$.

Conversely, let $x \in \textsc{Reach}(\textsc{Reach}(L, S), T)$. Then there are $z \in \textsc{Reach}(L, S)$ and $t \in T$ with $\bot \neq x = [\![t]\!](z)$. Additionally, there are $y \in L$ and $s \in S$ with $\bot \neq z = [\![s]\!](y)$ implying $\bot \neq x = [\![t]\!](z) = [\![t]\!]([\![s]\!](y)) = [\![st]\!](y)$. Hence, we infer $x \in \textsc{Reach}(L, ST)$.

Now, we prove (2). To this end, let $x \in \textsc{BackReach}(L, ST)$. Then we have $[\![ST]\!](x) \cap L \neq \varnothing$, i.e., there are $s \in S$, $t \in T$, and $y \in L$ with $\bot \neq y = [\![st]\!](x)$. Then there is $z \in U \setminus \{\bot\}$ with $[\![s]\!](x) = z$ and $[\![t]\!](z) = y$. Hence, we infer $z \in \textsc{BackReach}(L, T)$ and $x \in \textsc{BackReach}(\{z\}, S) \subseteq \textsc{BackReach}(\textsc{BackReach}(L, T), S)$.

Towards the converse inclusion, let $x \in \textsc{BackReach}(\textsc{BackReach}(L, T), S)$. Then there are $s \in S$ and $z \in \textsc{BackReach}(L, T)$ with $[\![s]\!](x) = z \neq \bot$. Additionally, there are $y \in L$ and $t \in T$ with $[\![t]\!](z) = y \neq \bot$. This implies $\bot \neq y = [\![t]\!](z) = [\![t]\!]([\![s]\!](x)) = [\![st]\!](x)$, i.e., $x \in \textsc{BackReach}(L, ST)$. ◂

We can also find the following connection between $\text{post}^*_{\mathfrak{S}}$ and REACH as well as between $\text{pre}^*_{\mathfrak{S}}$ and BACKREACH: we obtain $\text{post}^*_{\mathfrak{S}}$ and $\text{pre}^*_{\mathfrak{S}}$ from REACH and BACKREACH, respectively, by application of all action sequences which lead from an initial control state to a final control state. Note that the set of action sequences in $\mathfrak{S}$ is regular. We can see this by understanding the transition relation of $\mathfrak{S}$ as one of an extended NFA (recall that this is an NFA having edges labeled with words from Σ^*).

Observation 8.2.4. *Let $\mathcal{D}$ be a data type, $\mathfrak{S} = (Q, \mathcal{D}, \Delta, \Lambda)$ be a $\mathcal{D}$-system, and $C \subseteq \text{Conf}_{\mathfrak{S}} = Q \times U$ be a set of configurations of $\mathfrak{S}$. Let $\mathfrak{A} = (Q, \Sigma, Q, \Delta, Q)$ be an extended NFA derived from $\mathfrak{S}$ (note that Δ is the transition relation from $\mathfrak{S}$). Then we have:*

(1) $\text{post}^*_{\mathfrak{S}}(C) = \bigcup_{\iota, f \in Q} \{f\} \times \text{REACH}(\iota\, C, L(\mathfrak{A}_{\iota \to f}))$.

(2) $\text{pre}^*_{\mathfrak{S}}(C) = \bigcup_{\iota, f \in Q} \{\iota\} \times \text{BACKREACH}(f\, C, L(\mathfrak{A}_{\iota \to f}))$. ◂

8.3 Temporal Logics

Now, we want to recall some temporal logics on our Kripke-structures. With the help of these logics we are able to formalize properties of paths in our Kripke-structures. In verification this is used to specify the expected behavior of a given system. So, these logics help to check whether a system (or at least its formal model) satisfies a set of requirements specified by formulas from an adequate temporal logic. Possibly the most famous temporal logics are the *linear temporal logic* (LTL, for short, in some papers also called *propositional temporal logic* or PTL), the *computation tree logic* (CTL, for short), and their combination CTL*. The main difference between these logics is the set of allowed temporal operators. So, in the following we want to recall the definitions of some important operators. To this end, let $\mathcal{M} = (V, E, \Lambda)$ be a Kripke-structure on the atomic propositions AP. Additionally, let $\rho = (v_i)_{i \in \mathbb{N}}$ be a path in $\mathcal{M}$ and ϕ and ψ be two formulas. Then we define:

- $(\mathcal{M}, \rho) \vDash \top$,
- $(\mathcal{M}, \rho) \vDash p \in \text{AP}$ if, and only if, $p \in \Lambda(v_0)$,
- $(\mathcal{M}, \rho) \vDash \neg\phi$ if, and only if, $(\mathcal{M}, \rho) \nvDash \phi$,
- $(\mathcal{M}, \rho) \vDash \phi \vee \psi$ if, and only if, $(\mathcal{M}, \rho) \vDash \phi$ or $(\mathcal{M}, \rho) \vDash \psi$,
- $(\mathcal{M}, \rho) \vDash \text{X}\,\phi$ if, and only if, $(\mathcal{M}, \rho^1) \vDash \phi$,
- $(\mathcal{M}, \rho) \vDash \phi\,\text{U}\,\psi$ if, and only if, there is $k \in \mathbb{N}$ with $(\mathcal{M}, \rho^k) \vDash \psi$ and $(\mathcal{M}, \rho^\ell) \vDash \phi$ for each $0 \leq \ell < k$, and
- $(\mathcal{M}, \rho) \vDash \text{E}\,\phi$ if, and only if, there is a path $\sigma = (w_i)_{i \in \mathbb{N}}$ in $\mathcal{M}$ with $w_0 = v_0$ and $(\mathcal{M}, \sigma) \vDash \phi$.

In this connection we can read the formula "$\text{X}\,\phi$" as "ϕ holds in the ne**X**t step", "$\phi\,\text{U}\,\psi$" as "ϕ holds **U**ntil ψ holds", and "$\text{E}\,\phi$" as "there **E**xists a path satisfying ϕ".

The definitions of a few more temporal operators can be found in the literature (see, e.g., [GK07]). However, from the operators listed above we also obtain the following well-known operators:

- $\mathrm{F}\,\phi := \top\,\mathrm{U}\,\phi$ which reads as "Eventually (or **F**inally) ϕ holds."
- $\mathrm{G}\,\phi := \neg\,\mathrm{F}\,\neg\phi$ which reads as "**G**lobally the formula ϕ holds."
- $\mathrm{A}\,\phi := \neg\,\mathrm{E}\,\neg\phi$ which reads as "ϕ holds on **A**ll paths."

Then LTL consists of all of those formulas generated by Boolean operators and the temporal operators X and U (including F and G). The logic CTL consists of the Boolean operators and the combinations E X, E G, and E U of temporal operators. Finally, CTL^* is the set of all formulas generated by all of the operators listed above.

The *model checking problem* of these logics is defined as follows:

Input: A Kripke-structure $\mathcal{M} = (V, E, \Lambda)$, a node $v \in V$, and a formula ϕ
Question: Does $(\mathcal{M}, \rho) \vDash \phi$ hold for a path $\rho = (v_i)_{i\in\mathbb{N}}$ in $\mathcal{M}$ with $v_0 = v$?

Note that in the literature the model checking problem of LTL often is the following question: does $(\mathcal{M}, \rho) \vDash \phi$ hold for <u>all</u> paths ρ in $\mathcal{M}$ starting in node v? This is essentially the complementary problem of the problem defined above since $(\mathcal{M}, \rho) \vDash \phi$ holds for all paths ρ if, and only if, there is no path ρ with $(\mathcal{M}, \rho) \vDash \neg\phi$.

Now, recall the four problems introduced in the previous section. We can see that all questions correspond to instances of the model checking of LTL:

(1) To solve the reachability problem, we can check whether $\mathrm{F}\,\phi$ holds for a suitable formula ϕ.

(2) The safety problem asks whether $\mathrm{G}\,\phi$ holds for a formula ϕ.

(3) We can decide the liveness problem with the help of the formula $\mathrm{G}\,\mathrm{F}\,\phi$ where ϕ is an eligible formula.

(4) We can solve the inevitability problem by checking for satisfaction of $\phi\,\mathrm{U}\,\psi$ for two formulas ϕ and ψ.

In the next chapter we want to verify Kripke-structures induced by pushdown automata with one or more stacks. In other words, our aim is to prove that model checking for Kripke-structures $\mathcal{M}(\mathfrak{S})$ is decidable for some (multi-)pushdown systems $\mathfrak{S}$. The classical proof strategy to show the decidability of the model checking problem for LTL-like temporal logics (see, e.g., [VW86]) is as follows: let ϕ be a formula from a temporal logic, $\mathcal{M}(\mathfrak{S}) = (V, E, \Lambda)$ be a Kripke-structure induced by a $\mathcal{D}$-system $\mathfrak{S}$, and $v \in V$ be a node. We construct a *Büchi-automaton* (this is an NFA with different semantics) $\mathfrak{A} = (Q, \Gamma, I, \Delta, F)$ with the following properties:

- $\Gamma := 2^{\mathrm{cl}(\phi)}$ where $\mathrm{cl}(\phi)$ is the set of all subformulas of ϕ,
- $Q = F \subseteq 2^{\mathrm{cl}(\phi)}$ contains exactly the consistent sets of subformulas of ϕ,
- $I \subseteq 2^{\mathrm{cl}(\phi)}$ contains exactly the consistent sets of subformulas of ϕ which also contain ϕ, and
- Δ is a transition relation according to the semantics of the considered logic. Concretely, we have $(p, a, q) \in \Delta$ if $p = a$ and some further properties holds, e.g., if $\mathrm{X}\,\psi \in p \iff \psi \in q$ holds for any $\mathrm{X}\,\psi \in \mathrm{cl}(\phi)$.

Then we know ϕ is satisfiable by at least one Kripke-structure if the accepted ω-language of $\mathfrak{A}$ is not empty (note that $\mathfrak{A}$ does not depend on $\mathcal{M}$), i.e., if

$$L^\omega(\mathfrak{A}) := \{\alpha_0\alpha_1\alpha_2\ldots \in \Gamma^\omega \mid \exists (v_i)_{i\in\mathbb{N}} \colon v_0 \in I, \forall i \in \mathbb{N} \colon v_i \xrightarrow{\alpha_i}_{\mathfrak{A}} v_{i+1}, \exists^\infty j \in \mathbb{N} \colon v_j \in F\} \neq \varnothing$$

holds. Finally, we construct a (Büchi-)$\mathcal{D}$-automaton $\mathfrak{B}$ which simulates $\mathfrak{A}$ and $\mathfrak{S}$ in parallel. This means, the resulting automaton's states are tuples (p, q) of states from $\mathfrak{A}$ and $\mathfrak{S}$, in which the labeling $\Lambda(q)$ of q coincides with the atomic propositions $p \cap \mathrm{AP}$ in $p \subseteq \mathrm{cl}(\phi)$. We have an edge from one tuple to another one if, and only if, there are such edges in $\mathfrak{A}$ and in $\mathfrak{S}$. Then the formula ϕ holds in $\mathcal{M}(\mathfrak{S})$ on configuration v if, and only if, $\mathfrak{B}$ accepts at least one ω-word, i.e., if the following ω-language is not empty:

$$L^\omega(\mathfrak{B}) := \{\alpha_0\alpha_1\alpha_2\ldots \in \Gamma^\omega \mid \exists (c_i)_{i\in\mathbb{N}} \colon c_0 \in \mathrm{Init}_{\mathfrak{B}}, \forall i \in \mathbb{N} \colon c_i \xrightarrow{\alpha_i}_{\mathfrak{B}} c_{i+1}, \exists^\infty j \in \mathbb{N} \colon c_j \in \mathrm{Final}_{\mathfrak{B}}\}.$$

We also call the emptiness problem of Büchi-$\mathcal{D}$-automata the *recurrent reachability problem* (since such automaton accepts an ω-word iff it reaches an accepting state infinitely often).

Theorem 8.3.1 ([VW86]). *Let $\mathcal{D}$ be a data type. The model checking problem for several temporal logics (like* LTL*) on Kripke-structures induced from $\mathcal{D}$-systems can be reduced to the emptiness problem of Büchi-$\mathcal{D}$-automata. In this reduction we construct a Büchi-$\mathcal{D}$-automaton in time exponential in the size of ϕ and which has exponentially many states.* ◂

In other words, to decide the model checking problem for LTL and other temporal logics, we only have to prove that the emptiness problem in Büchi-$\mathcal{D}$-automata is decidable. For example, the emptiness in Büchi-automata is NL-complete. Hence, we obtain the decidability of the model checking problem for finite Kripke-structures:

Corollary 8.3.2 ([Pnu77, VW86]). *The model checking problem of* LTL *is* PSPACE*-complete for finite Kripke-structures.* ◂

Note that we can also show that the model checking problem of CTL for finite Kripke-structures is in P (however this result does not use the aforementioned reduction [CE82]) and the model checking problem of CTL* also is PSPACE-complete [EH83]. In the next chapter we will use this reduction to prove the decidability of the model checking problem for Kripke-structures induced by pushdown automata with one or more stacks.

Chapter 9

Verifying Pushdown Automata

9.1 Introduction

In this chapter we want to verify pushdown systems and automata having one or more stacks. Such systems are often used to model programs in functional programming languages (like Erlang, Haskell, or Scala), in which one recursively calls a finite number of functions with finite domains.

It is well-known that the reachability problem in pushdown systems with one such stack is decidable in polynomial time [BEM97]. Moreover, in such systems the model checking problems of several temporal logics are decidable. For example, Bouajjani et al. proved in [BEM97] the EXPTIME-completeness of the model checking problem for LTL. Walukiewicz showed that additionally model checking for CTL and the so-called *propositional μ-calculus* are EXPTIME-complete while model checking of the EF-fragment of CTL is PSPACE-complete [Walo0, Walo1]. Moreover, we know that CTL^*-model checking is 2EXPTIME-complete [Bozo7]. Hence, with these results we are able to analyze such aforementioned functional programs with finite domains towards their correctness and other interesting properties.

We can also consider pushdown systems having at least two stacks. With the help of such systems we can model functional programs with finite domains within concurrent or distributed systems. Indeed, such systems are known to be as powerful as Turing-machines. Hence, the reachability problem and the model checking problem for any temporal logic gets undecidable in such pushdown systems. Here, we will consider a special restriction of such multi-pushdown systems and prove the decidability of the reachability and recurrent reachability problem implying the decidability of the model checking problem of LTL and other temporal logics.

But first, we recall the results and constructions for pushdown systems with one stack.

9.2 Reliable Stacks

In this section we consider the reachability problem in pushdown automata having one stack. It is a very famous result that this problem is decidable in polynomial time. However, we want to recall some constructions proving this result. Later in this chapter we will generalize two of these constructions to automata having multiple reliable stacks.

First, we need an appropriate method to store sets of configurations of a PDA. To this end, let $\mathfrak{S} = (Q, \mathcal{P}_A, \Delta, \Lambda)$ be a $\mathcal{P}_A$-system (we also call these systems *pushdown systems*).

An $\mathfrak{S}$-*NFA* is an NFA $\mathfrak{A} = (S, A, Q, \delta, F)$ with $Q \subseteq S$. The *accepted configurations* of the $\mathfrak{S}$-NFA $\mathfrak{A} = (S, A, Q, \delta, F)$ are

$$C(\mathfrak{A}) := \{(q, w) \mid q \in Q, q \xrightarrow{w}_{\mathfrak{A}} F\}.$$

A set of configurations $C \subseteq \mathrm{Conf}_{\mathfrak{S}}$ is called *regular* if there is an $\mathfrak{S}$-NFA $\mathfrak{A}$ with $C(\mathfrak{A}) = C$.

Note that we can see the set of accepted configurations of an $\mathfrak{S}$-NFA $\mathfrak{A}$ as a collection of a finite number of regular languages, each starting from a state of $\mathfrak{S}$. In other words, for $\mathfrak{S} = (Q, \mathcal{P}_A, \Delta, \Lambda)$ and $\mathfrak{A} = (S, A, Q, \delta, F)$ we have the following equation:

$$C(\mathfrak{A}) = \bigcup_{q \in Q} \{q\} \times L(\mathfrak{A}_{q \to F}).$$

Hence, the term "regular set of configurations" is eligible.

9.2.1 Reachability

Now, we are able to state the main theorem of this section: the decidability of the reachability problem in pushdown systems:

Theorem 9.2.1 ([BEM97, FWW97, Esp+00]). *Let A be an alphabet, $\mathfrak{S}$ be a $\mathcal{P}_A$-system, $C \subseteq \mathrm{Conf}_{\mathfrak{S}}$ be regular. Then $\mathrm{post}^*_{\mathfrak{S}}(C)$ and $\mathrm{pre}^*_{\mathfrak{S}}(C)$ are regular. In particular, we can compute $\mathfrak{S}$-NFAs accepting $\mathrm{post}^*_{\mathfrak{S}}(C)$ and $\mathrm{pre}^*_{\mathfrak{S}}(C)$ from $\mathfrak{S}$ and an $\mathfrak{S}$-NFA accepting C in polynomial time.*

We recall multiple constructions proving this result. We also want to visualize the proof ideas with the help of an example. To this end, consider the following pushdown system $\mathfrak{S}$ and $\mathfrak{S}$-NFA $\mathfrak{A}$:

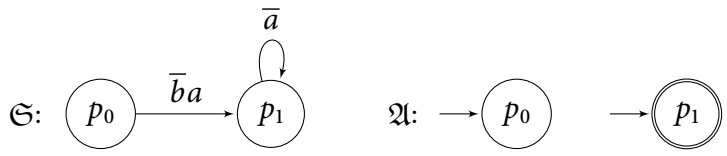

Figure 9.1

The first construction we consider is the one introduced by Bouajjani, Esparza, and Maler [BEM97]. In this construction we only consider classical pushdown systems having transitions labeled with actions sequences from $\overline{A}A^*$, only, i.e., on each step the PDA reads one letter and afterwards writes a word into the stack. However, as we stated earlier in this thesis, any pushdown system can be translated into such classical pushdown system.

Proof idea. Let $\mathfrak{S} = (Q, \mathcal{P}_A, \Delta, \Lambda)$ where for each $(p, t, q) \in \Delta$ we have $t \in \overline{A}A^*$ and let C be accepted by $\mathfrak{A} = (S, A, Q, \delta, F)$. We iteratively add some edges to $\mathfrak{A}$ simulating the application of a single transition of $\mathfrak{S}$. Since $\mathfrak{A}$ is finite, this construction terminates after several iterations. The number of these iterations is polynomial in the size of $\mathfrak{A}$.

To this end, we define $\mathfrak{S}$-NFAs $\mathfrak{A}^{(i)} := (S, A, Q, \delta^{(i)}, F)$ for numbers $i \in \mathbb{N}$ as follows:

- Set $\delta^{(0)} := \delta$, i.e., we have $\mathfrak{A}^{(0)} = \mathfrak{A}$. W.l.o.g., we assume that $\delta^{(0)} \subseteq S \times A \times S \setminus Q$ holds, i.e., the initial states of $\mathfrak{A}^{(0)}$ have no in-edges.

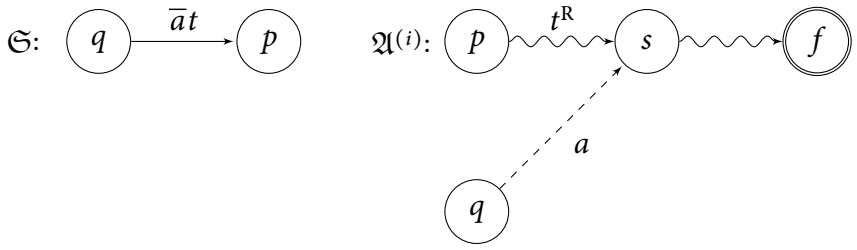

Figure 9.2. Visualization of the construction. The dashed a-edge is added in step $i+1$.

- $\delta^{(i+1)} := \delta^{(i)} \cup \{(q, a, s) \in Q \times A \times S \mid \exists (q, \overline{a}t, p) \in \Delta \colon p \xrightarrow{t^{\mathrm{R}}}_{\mathfrak{A}^{(i)}} s\}$. In other words, whenever there is a transition $(q, \overline{a}t, p)$ in $\mathfrak{S}$ and a configuration $(p, t^{\mathrm{R}}w) \in C(\mathfrak{A}^{(i)})$ we also have $(q, aw) \in \mathrm{pre}^1_{\mathfrak{S}}(C(\mathfrak{A}^{(i)})) \subseteq \mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{A}))$. Hence, we add an a-labeled edge from q in $\mathfrak{A}^{(i+1)}$ (cf. Figure 9.2).

Let $\mathfrak{A}^{(\infty)} = (S, A, Q, \delta^{(\infty)}, F)$ be the "limit" of the sequence $(\mathfrak{A}^{(i)})_{i \in \mathbb{N}}$, i.e., $\delta^{(\infty)} = \bigcup_{i \in \mathbb{N}} \delta^{(i)}$. Then we can show $\mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{A})) = C(\mathfrak{A}^{(\infty)})$. Since we have $\delta^{(0)} \subseteq \delta^{(1)} \subseteq \cdots \subseteq \delta^{(\infty)} \subseteq S \times A \times S$ and since the latter set is finite, we have $\delta^{(k)} = \delta^{(k+1)}$ for $k := |S|^2|A|$ and, therefore $\delta^{(k)} = \delta^{(\infty)}$. Hence, $\mathfrak{A}^{(\infty)}$ also is constructible from $\mathfrak{S}$ and $\mathfrak{A}$ in polynomial time.◂

Example 9.2.2. Consider $\mathfrak{S}$ and $\mathfrak{A}$ as visualized in Figure 9.1. Then we have $C(\mathfrak{A}) = \{(p_1, \varepsilon)\}$ and we want to compute $\mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{A}))$. In the first approach we compute an $\mathfrak{S}$-NFA as depicted in Figure 9.3.

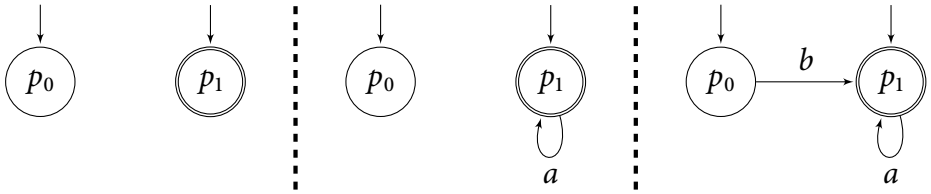

Figure 9.3. The $\mathfrak{S}$-NFAs $\mathfrak{A}^{(0)}$, $\mathfrak{A}^{(1)}$, and $\mathfrak{A}^{(2)}$ (from left to right). The sequence $(\mathfrak{A}^{(i)})_{i \in \mathbb{N}}$ stabilizes after the second step. Hence, we have $\mathfrak{A}^{(\infty)} = \mathfrak{A}^{(2)}$.

Hence, we have $\mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{A})) = C(\mathfrak{A}^{(2)}) = (\{p_0\} \times ba^*) \cup (\{p_1\} \times a^*)$. ⌟

Now, we take a closer look at the complexity of this first construction. We see that the resulting $\mathfrak{S}$-NFA $\mathfrak{A}^{(\infty)}$ has still as many states as $\mathfrak{A}$, but more transitions. However, the number of transitions is bounded by $O(|\mathfrak{A}|^2 \cdot |A|)$. To add a new transition to $\mathfrak{A}$, we have to find for any pair of states in $\mathfrak{S}$ a path of bounded length $\ell := \max\{|t| \mid (q, \overline{a}t, p) \in \Delta\}$ in $\mathfrak{A}$. Hence, each round is possible in time $O(\ell \cdot |Q|^2)$ implying a total running time bounded by $O(|\mathfrak{S}|^3 \cdot |\mathfrak{A}|^2 \cdot |A|)$ (since $\ell, |Q| \in O(|\mathfrak{S}|)$).

However, this approach is not convenient for computing the succeeding configurations of $C(\mathfrak{A})$. This is due to the restriction that each transition of $\mathfrak{S}$ is labeled with a word from $\overline{A}A^*$. Computing the forwards reachable configurations would require to add some states to

$\mathfrak{A}$. Since we may add new states in each round, the construction will possibly never terminate. Though, it is possible to modify this construction as follows:

(1) We can translate the pushdown system into a *reversible* one in which each transition is labeled with an action sequence of the form $\overline{a}$ (decrease the stack), $\overline{a}b$ (modify the top symbol), or $\overline{a}ab$ (push another symbol onto the stack). This can be done by splitting transitions as described by Schellmann in [Sch19]: a transition labeled with $\overline{a}b_1b_2\dots b_n$ (for $n \geq 2$) of $\mathfrak{S}$ can be split into multiple transitions labeled with $\overline{a}b_1$ and $\overline{b_i}b_ib_{i+1}$ for each $1 \leq i < n$.

For such reversible pushdown system $\mathfrak{S}$ we can construct the *reverse* pushdown system $\mathfrak{T}$ as follows: replace $(p, \overline{a}b, q)$ by $(q, \overline{b}a, p)$, $(p, \overline{a}ab, q)$ is replaced by $(q, \overline{b}, p)$, and $(p, \overline{a}, q)$ is replaced by $(q, \overline{b}ba, p)$. Then we have $c \to_{\mathfrak{S}}^* d$ if, and only if, $d \to_{\mathfrak{T}}^* c$ holds for any two configurations $c, d \in \mathrm{Conf}_{\mathfrak{S}} = \mathrm{Conf}_{\mathfrak{T}}$. Finally, we can apply the construction from [BEM97] as described above to compute the forwards reachable configurations (cf. [Sch19]).

(2) An alternative algorithm is described in [Esp+00]. In that paper the authors added a fixed number of states to $\mathfrak{A}$ representing the long tail w of a transition labeled with $\overline{a}bw$. Afterwards, we obtain an algorithm which is very similar to the one from [BEM97].

Another construction suitable for computing the sets of forwards and backwards reachable configurations was presented by Finkel, Willems, and Wolper in [FWW97]:

Proof idea. Let $\mathfrak{S} = (Q, \mathcal{P}_A, \Delta, \Lambda)$ be a pushdown system such that $(p, t, q) \in \Delta$ implies $t \in \overline{A}A^*$ (i.e., $\mathfrak{S}$ is a classical pushdown system). Suppose there are the following two transitions $(p, \overline{a}tb, q), (q, \overline{b}, r) \in \Delta$ implying $(p, aw) \to_{\mathfrak{S}} (q, bt^{\mathrm{R}}w) \to_{\mathfrak{S}} (r, t^{\mathrm{R}}w)$ for any word $w \in A^*$. If we add the transition $(p, \overline{a}t, r)$ to Δ that allows to go from (p, aw) to $(r, t^{\mathrm{R}}w)$ directly, the reachability relation does not change. We keep adding such "shortcuts" and call the resulting pushdown system $\mathfrak{S}^{(\infty)}$ *saturated*. Then, any run of the original system $\mathfrak{S}$ can be simulated by a run of the saturated system $\mathfrak{S}^{(\infty)}$ that first shortens the stack and then writes onto the stack. It follows that for such saturated pushdown systems the mappings $\mathrm{post}^*_{\mathfrak{S}}$ and $\mathrm{pre}^*_{\mathfrak{S}}$ preserve regularity effectively.

The crucial point of this construction is that any run of the system $\mathfrak{S}^{(\infty)}$ can be brought into such "simple form" using the shortcuts. Here, "simple form" means that a run consists of two phases: the pushdown decreases properly in every step of the first phase and does not decrease in any step of the second phase.

Note that this result agrees with the fact that any action sequence $t \in \Sigma^*$ of a pushdown behaves equivalently to an action sequence from $\overline{A}^* A^*$ (or $\perp\!\!\!\perp$), cf. Theorem 4.3.2. ◂

Example 9.2.3. Again, we consider the pushdown system $\mathfrak{S}$ and $\mathfrak{S}$-NFA $\mathfrak{A}$ as displayed in Figure 9.1. Since we have $(p_0, \overline{b}a, p_1), (p_1, \overline{a}, p_1) \in \Delta$, the equivalent saturated pushdown system $\mathfrak{S}^{(\infty)}$ also has a transition $(p_0, \overline{b}, p_1)$.

Then we see that $\mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{A}))$ is the following set:

$$\big(\{p_0\} \times ba^*\big) \cup \big(\{p_1\} \times a^*\big),$$

which is in fact regular (however, the proof idea from above does not describe, how to construct an accepting $\mathfrak{A}$-NFA). ⌟

Finally, we have to consider the complexity of this sketched construction. First, consider the construction of the equivalent saturated pushdown system $\mathfrak{S}^{(\infty)}$. We can compute this

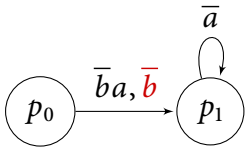

Figure 9.4. The saturated pushdown system $\mathfrak{S}^{(\infty)}$.

system iteratively by adding new transitions having a label shorter than the previously existing ones. Hence, our algorithm requires at most $O(|Q|^2 \cdot |t|)$ where $t \in A^*$ is the longest word with $(p, \overline{a}t, q) \in \Delta$. According to [FWW97] it is also possible to compute an $\mathfrak{S}$-NFA accepting $\mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{A}))$ in cubic time.

From Theorem 9.2.1 we finally obtain the following decidability result:

Corollary 9.2.4. *The following reachability problem is decidable in polynomial time:*

Input: *An alphabet A, a $\mathcal{P}_A$-system $\mathfrak{S}$, and two $\mathfrak{S}$-NFAs $\mathfrak{A}$ and $\mathfrak{B}$.*
Question: *Does $C(\mathfrak{A}) \to^*_{\mathfrak{S}} C(\mathfrak{B})$ hold?* ◂

9.2.2 Recurrent Reachability

Using the decidability of the reachability problem in pushdown systems, we can also prove that repeated reachability is decidable. This repeated reachability problem corresponds to the emptiness problem of ω-languages accepted by Büchi-pushdown automata. Applying the reduction in Theorem 8.3.1 we obtain the decidability of the model checking problem for several temporal logics in pushdown systems.

Lemma 9.2.5 ([BEM97]). *Let A be an alphabet, $\mathfrak{A} = (Q, \Gamma, \mathcal{P}_A, I, c, \Delta, F)$ be a pushdown automaton. Then we have $L^\omega(\mathfrak{A}) \neq \varnothing$ if, and only if, there are $q \in Q$ and $a \in A$ with*

(1) $\mathrm{pre}^*_{\mathfrak{A}}(\{q\} \times aA^*) \cap \mathrm{Init}_{\mathfrak{A}} \neq \varnothing$ *and*
(2) $(q, a) \in \mathrm{pre}^+_{\mathfrak{A}}(\mathrm{Final}_{\mathfrak{A}} \cap \mathrm{pre}^+_{\mathfrak{A}}(\{q\} \times aA^*))$. ◂

Note that both conditions in Lemma 9.2.5 are decidable in polynomial time due to Corollary 9.2.4. Hence, we can see that the emptiness problem for ω-languages accepted by Büchi-PDAs is decidable in polynomial time.

Now, we may consider the model checking problem of LTL and other temporal logics. We are able to reduce model checking for LTL-like temporal logics to the emptiness problem of Büchi-automata. Hence, from Lemma 9.2.5 we obtain that the model checking problem is decidable for Kripke-structures induced by pushdown systems in this case. Concretely, we know that model checking of LTL is EXPTIME-complete for this kind of Kripke-structures

[BEM97]. We also know that CTL-model checking is EXPTIME-complete [Walo0, Walo1] and CTL*-model checking is 2EXPTIME-complete [Bozo7].

9.3 Partially Lossy Stacks

Next, we want to transfer the results from the previous section to automata having one partially lossy stack. Recall that we have learned two different semantics of partially lossy stacks which both have the same computational power. However, in the context of forwards and backwards reachable configurations we have to differentiate between these two types. We first show that we obtain the configurations reachable in a partially lossy stack system with default semantics from one with read-lossy semantics. To this end, we have to extend the notion of $\mathcal{L}$-subwords from words to configurations. So, let $\mathfrak{S} = (Q, \mathcal{P}_\mathcal{L}, \Delta, \Lambda)$ be a $\mathcal{P}_\mathcal{L}$-system and $(p, v), (q, w) \in \mathrm{Conf}_\mathfrak{S}$ be two configurations. We write $(p, v) \sqsubseteq_\mathcal{L} (q, w)$ if, and only if, $p = q$ and $v \sqsubseteq_\mathcal{L} w$ holds. In other words, we have $c \sqsubseteq_\mathcal{L} d$ for two configurations $c, d \in \mathrm{Conf}_\mathfrak{S}$ if c and d agree in their control state and the content in c is an $\mathcal{L}$-subword of the one in d. Obviously, the relation $\sqsubseteq_\mathcal{L}$ is a partial ordering on $\mathrm{Conf}_\mathfrak{S}$.

Now, let $\mathfrak{S} = (Q, \mathcal{PLS}_\mathcal{L}, \Delta, \Lambda)$ be a partially lossy stack system with default semantics. From this system we obtain a pls system $\mathfrak{T} = (Q, \mathcal{P}_\mathcal{L}, \Delta, \Lambda)$ with read-lossy semantics simply by replacing the underlying data type. Note that this system is still well-defined. Since $\mathcal{P}_\mathcal{L}$ is only a determinization of $\mathcal{PLS}_\mathcal{L}$, the Kripke-structure $\mathcal{M}(\mathfrak{T})$ is a subgraph of $\mathcal{M}(\mathfrak{S})$. Hence, each run of $\mathfrak{T}$ is also a run in $\mathfrak{S}$. Conversely, there may be runs in $\mathfrak{S}$ which are not allowed in $\mathfrak{T}$. However, for each run $c \to_\mathfrak{S}^* d$ in $\mathfrak{S}$ we can find another run in $\mathfrak{T}$ ending in a configuration d' with $d \sqsubseteq_\mathcal{L} d'$, i.e., in which its stack content is an $\mathcal{L}$-superword of the one in d. This is the following lemma:

Lemma 9.3.1. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $\mathfrak{S} = (Q, \mathcal{PLS}_\mathcal{L}, \Delta, \Lambda)$ be a $\mathcal{PLS}_\mathcal{L}$-system, and $\mathfrak{T} = (Q, \mathcal{P}_\mathcal{L}, \Delta, \Lambda)$ be its read-lossy version. For any pair $c, d \in \mathrm{Conf}_\mathfrak{S} = \mathrm{Conf}_\mathfrak{T}$ the following equivalence holds:*

$$c \to_\mathfrak{S}^* d \iff \text{there is } d' \in \mathrm{Conf}_\mathfrak{T} \colon d \sqsubseteq_\mathcal{L} d' \text{ and } c \to_\mathfrak{T}^* d' .$$

Proof. First we prove the implication "$\Leftarrow$". It is easy to see that $\mathcal{M}(\mathfrak{T})$ is a subgraph of $\mathcal{M}(\mathfrak{S})$. Hence, from $c \to_\mathfrak{T}^* d'$ we also obtain $c \to_\mathfrak{S}^* d'$. Additionally, from $d \sqsubseteq_\mathcal{L} d'$ we also obtain an ε-edge in $\mathcal{M}(\mathfrak{S})$ implying $c \to_\mathfrak{S}^* d' \to_\mathfrak{S} d$.

Now, we prove the converse implication. To this end, we prove by induction on the length $n \in \mathbb{N}$ of a run $c \to_\mathfrak{S}^n d$ that there is $d' \in \mathrm{Conf}_\mathfrak{T}$ with $d \sqsubseteq_\mathcal{L} d'$ and $c \to_\mathfrak{T}^* d'$ (note that this run has not necessarily length n). If $n = 0$ holds, we have $c = d$ and $d \sqsubseteq_\mathcal{L} d$ (recall that $\sqsubseteq_\mathcal{L}$ is a partial ordering). Hence, setting $d' := d$ yields our claim in this case. Next, let $n \geq 1$. Then there is $e \in \mathrm{Conf}_\mathfrak{S}$ with $c \to_\mathfrak{S}^{n-1} e \to_\mathfrak{S} d$. By induction hypothesis there is a configuration $e' \in \mathrm{Conf}_\mathfrak{T}$ with $e \sqsubseteq_\mathcal{L} e'$ and $c \to_\mathfrak{T}^* e'$. Let $e := (p, v)$, $e' := (p, v')$, and $d := (q, w)$. We have to distinguish two cases:

(1) $p = q$ and $w \sqsubseteq_\mathcal{L} v$ (i.e., $e \to_\mathfrak{S} d$ only forgets some letters). Then we have $d \sqsubseteq_\mathcal{L} e \sqsubseteq_\mathcal{L} e'$. By transitivity of $\sqsubseteq_\mathcal{L}$ we are done with $d' := e'$.

(2) There is $(p, t, q) \in \Delta$ with $w \in [\![t]\!]_{\mathcal{PLS}_\mathcal{L}}(v)$. By Proposition 3.3.4 we have $w \sqsubseteq_\mathcal{L} [\![t]\!]_{\mathcal{P}_\mathcal{L}}(v)$ and from Lemma 3.3.3 we learn $[\![t]\!]_{\mathcal{P}_\mathcal{L}}(v) \sqsubseteq_\mathcal{L} [\![t]\!]_{\mathcal{P}_\mathcal{L}}(v')$. Set $d' := (q, [\![t]\!]_{\mathcal{P}_\mathcal{L}}(v'))$. Then we have $d \sqsubseteq_\mathcal{L} d'$ by transitivity of $\sqsubseteq_\mathcal{L}$. We also have $e' \to_\mathfrak{T} d'$ implying $c \to_\mathfrak{T}^* e' \to_\mathfrak{T} d'$. ◂

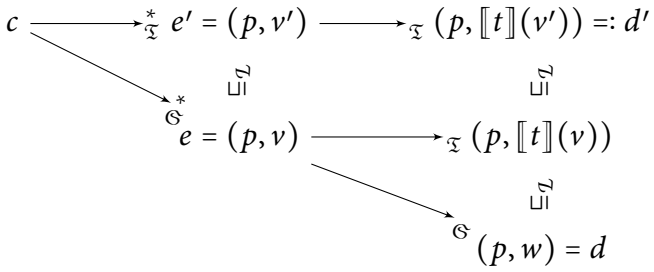

Figure 9.5. Visualization of the proof of Lemma 9.3.1.

From this result we learn the following connection between forwards resp. backwards reachable configurations in a partially lossy stack system with default and read-lossy semantics.

Proposition 9.3.2. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $\mathfrak{S} = (Q, \mathcal{PLS}_\mathcal{L}, \Delta, \Lambda)$ be a $\mathcal{PLS}_\mathcal{L}$-system, $\mathfrak{T} = (Q, \mathcal{P}_\mathcal{L}, \Delta, \Lambda)$ be its read-lossy version, and $C \subseteq \mathrm{Conf}_\mathfrak{S}$ be a set of configurations. Then the following statements hold:*

(1) $\mathrm{post}^*_\mathfrak{S}(C) = {\downarrow_{\sqsubseteq_\mathcal{L}}} \mathrm{post}^*_\mathfrak{T}(C)$ *and*

(2) $\mathrm{pre}^*_\mathfrak{S}(C) = \mathrm{pre}^*_\mathfrak{T}({\uparrow_{\sqsubseteq_\mathcal{L}}} C)$.

Proof. First we prove (1). Let $d \in \mathrm{post}^*_\mathfrak{S}(C)$. Then there is $c \in C$ with $c \to^*_\mathfrak{S} d$. By Lemma 9.3.1 there is $d' \in \mathrm{Conf}_\mathfrak{T}$ with $d \sqsubseteq_\mathcal{L} d'$ and $c \to^*_\mathfrak{T} d'$. Hence, we have $d' \in \mathrm{post}^*_\mathfrak{T}(C)$ and, therefore, $d \in {\downarrow_{\sqsubseteq_\mathcal{L}}} \mathrm{post}^*_\mathfrak{T}(C)$.

Conversely, let $d \in {\downarrow_{\sqsubseteq_\mathcal{L}}} \mathrm{post}^*_\mathfrak{T}(C)$. Then there are $c \in C$ and $d' \in \mathrm{Conf}_\mathfrak{T}$ with $d \sqsubseteq_\mathcal{L} d'$ and $c \to^*_\mathfrak{T} d'$. According to Lemma 9.3.1 we learn $c \to^*_\mathfrak{S} d$ implying $d \in \mathrm{post}^*_\mathfrak{S}(C)$.

Now, we show the second statement. So, let $c \in \mathrm{pre}^*_\mathfrak{S}(C)$. Then there is $d \in C$ with $c \to^*_\mathfrak{S} d$. We find a configuration $d' \in \mathrm{Conf}_\mathfrak{T}$ with $d \sqsubseteq_\mathcal{L} d'$ and $c \to^*_\mathfrak{T} d'$ by Lemma 9.3.1. In other words, we have $d' \in {\uparrow_{\sqsubseteq_\mathcal{L}}} C$ implying $c \in \mathrm{pre}^*_\mathfrak{T}({\uparrow_{\sqsubseteq_\mathcal{L}}} C)$.

Finally, let $c \in \mathrm{pre}^*_\mathfrak{T}({\uparrow_{\sqsubseteq_\mathcal{L}}} C)$. Then there is $d' \in {\uparrow_{\sqsubseteq_\mathcal{L}}} C$ with $c \to^*_\mathfrak{T} d'$. Additionally, there is $d \in C$ with $d \sqsubseteq_\mathcal{L} d'$. Then by Lemma 9.3.1 we obtain $c \to^*_\mathfrak{S} d$ and, hence, $c \in \mathrm{pre}^*_\mathfrak{S}(C)$. ◂

Let $C \subseteq \mathrm{Conf}_\mathfrak{S}$ be a regular set of configurations. Then we can compute $\mathfrak{S}$-NFAs accepting ${\uparrow_{\sqsubseteq_\mathcal{L}}} C$ and ${\downarrow_{\sqsubseteq_\mathcal{L}}} C$, respectively, in polynomial time and linear space. The constructions are similar to the ones from Haines [Hai69]. In other words, if we restrict Proposition 9.3.2 to regular sets of configurations, we obtain the forwards and backwards reachable configurations of a pls system with default semantics from one with read-lossy semantics using these closure properties, only.

Next, we want to reduce the reachability problem of a partially lossy stack system to one of a reliable pushdown system. To this end, we construct from a $\mathcal{P}_\mathcal{L}$-system $\mathfrak{S}$ a $\mathcal{P}_A$-system $\mathfrak{T}$ which reaches the same configurations (and some further configurations not in $\mathrm{Conf}_\mathfrak{S}$).

Proposition 9.3.3. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $\mathfrak{S}$ be a $\mathcal{P}_\mathcal{L}$-system. Then we can compute a $\mathcal{P}_A$-system $\mathfrak{T}$ in polynomial time with $\mathrm{Conf}_\mathfrak{S} \subseteq \mathrm{Conf}_\mathfrak{T}$ such that we have for each $C \subseteq \mathrm{Conf}_\mathfrak{S}$:*

$$\mathrm{post}^*_\mathfrak{S}(C) = \mathrm{post}^*_\mathfrak{T}(C) \cap \mathrm{Conf}_\mathfrak{S} \quad \textit{and} \quad \mathrm{pre}^*_\mathfrak{S}(C) = \mathrm{pre}^*_\mathfrak{T}(C) \cap \mathrm{Conf}_\mathfrak{S}\,.$$

Proof idea. Let $\mathfrak{S} = (Q_\mathfrak{S}, \mathcal{P}_\mathcal{L}, \Delta_\mathfrak{S}, \Lambda_\mathfrak{S})$. Without loss of generality, we can assume that each transition $(p, t, q) \in \Delta_\mathfrak{S}$ applies at most one basic action (i.e., $|t| \leq 1$ holds). Otherwise, we are able to split transitions in $\mathfrak{S}$.

We construct a pushdown system $\mathfrak{T}$ which simulates the behavior of $\mathfrak{S}$. But whenever $\mathfrak{S}$ tries to read a letter, $\mathfrak{T}$ guesses this letter $a \in A$ (and stores its guess in its global state). In this new state we can read some other forgettable letters from $F \setminus \{a\}$. Finally, we read a and continue the simulation of $\mathfrak{S}$. Then $\mathfrak{T} = (Q_\mathfrak{T}, \mathcal{P}_A, \Delta_\mathfrak{T}, \Lambda_\mathfrak{T})$ is defined as follows:

- $Q_\mathfrak{T} := Q_\mathfrak{S} \cup (Q_\mathfrak{S} \times A)$,
- $\Lambda_\mathfrak{T}(q) = \Lambda_\mathfrak{S}(q)$ and $\Lambda_\mathfrak{T}((q, a)) = \Lambda_\mathfrak{S}(q)$ for each $q \in Q_\mathfrak{S}$ and $a \in A$, and
- $\Delta_\mathfrak{T}$ consists of the following transitions:
  - $(p, a, q) \in \Delta_\mathfrak{T}$ for each $a \in A \cup \{\varepsilon\}$ and $(p, a, q) \in \Delta_\mathfrak{S}$ (i.e., $\mathfrak{T}$ inherits the write and ε-transitions of $\mathfrak{S}$) and
  - $(p, \varepsilon, (p, a)), ((p, a), \overline{b}, (p, a)), ((p, a), \overline{a}, q) \in \Delta_\mathfrak{T}$ for each $(p, \overline{a}, q) \in \Delta_\mathfrak{S}$ and $b \in F \setminus \{a\}$ (i.e., $\mathfrak{T}$ simulates read actions from $\mathfrak{S}$ in three steps).

Then we can show

$$c \to^*_\mathfrak{S} d \iff c \to^*_\mathfrak{T} d$$

for each pair of configurations $c, d \in \mathrm{Conf}_\mathfrak{S}$ of $\mathfrak{S}$. This finally implies our claim. ◂

Combining Propositions 9.3.2 and 9.3.3 with the results concerning (reliable) pushdown systems from the previous section, we can also compute the forwards and backwards reachable configurations of a regular set of configurations of a partially lossy stack system with default or read-lossy semantics. Additionally, we obtain the decidability of the recurrent reachability problem in these cases:

Theorem 9.3.4. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet. Then the following statements hold:*

(1) Let $\mathfrak{S}$ be a $\mathcal{PLS}_\mathcal{L}$- or $\mathcal{P}_\mathcal{L}$-system and $C \subseteq \mathrm{Conf}_\mathfrak{S}$ be a regular set of configurations. Then we can compute $\mathfrak{S}$-NFAs accepting $\mathrm{post}^*_\mathfrak{S}(C)$ and $\mathrm{pre}^*_\mathfrak{S}(C)$ in polynomial time.

(2) Let $\mathfrak{A}$ be a $\mathcal{PLS}_\mathcal{L}$- or $\mathcal{P}_\mathcal{L}$-automaton. We can decide in polynomial time, whether $L^\omega(\mathfrak{A}) \neq \varnothing$ holds. ◂

As a corollary we also obtain the decidability of the model checking problem of temporal logics like LTL in such systems.

9.4 Intermezzo: Asynchronous Automata

Before verifying the distributed version of pushdown systems, we have to introduce some acceptors of rational and recognizable trace languages. To this end, we introduce the so-called asynchronous automata.

Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and let $\mathfrak{A} = (Q, A, I, \Delta, F)$ be an NFA. The *accepted trace language* of $\mathfrak{A}$ is

$$T(\mathfrak{A}) := \{[w] \mid I \xrightarrow{w}_{\mathfrak{A}} F\} = \zeta_{\mathcal{A}}(L(\mathfrak{A}))$$

Note that by $T(\mathfrak{A}) = \zeta_{\mathcal{A}}(L(\mathfrak{A}))$ we infer that the accepted trace language of $\mathfrak{A}$ is rational. Moreover, since $\zeta_{\mathcal{A}}$ is an epimorphism, NFAs accept exactly the class of rational trace languages.

Until now, NFAs accept a trace $\lambda \in \mathbb{M}(\mathcal{A})$ if there is at least one accepting run labeled with a serialization $w \in \lambda$. Next, we want to introduce restricted NFAs $\mathfrak{A}$ which accept a trace $\lambda \in \mathbb{M}(\mathcal{A})$ if they accept <u>all</u> words $w \in \lambda$. In this case, we obtain $L(\mathfrak{A}) = \zeta_{\mathcal{A}}^{-1}(\zeta_{\mathcal{A}}(L(\mathfrak{A})))$ implying that the accepted trace language $T(\mathfrak{A})$ is recognizable. This special restriction is the following one:

Definition 9.4.1. Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet. An *asynchronous automaton* is an NFA $\mathfrak{A} = (\vec{\boldsymbol{Q}}, A, I, \Delta, F)$ where $\vec{\boldsymbol{Q}} = \prod_{i \in P} Q_i$ is the product of some finite sets Q_i of *local states* and where for each transition $(\vec{\boldsymbol{p}}, a, \vec{\boldsymbol{q}}) \in \Delta$ we have the following restrictions:

(1) $\vec{\boldsymbol{p}}{\restriction}_{P \setminus a\,\mathrm{M}} = \vec{\boldsymbol{q}}{\restriction}_{P \setminus a\,\mathrm{M}}$ and

(2) for each $\vec{\boldsymbol{r}} \in \vec{\boldsymbol{Q}}{\restriction}_{P \setminus a\,\mathrm{M}}$ we have $((\vec{\boldsymbol{p}}{\restriction}_{a\,\mathrm{M}}, \vec{\boldsymbol{r}}), a, (\vec{\boldsymbol{q}}{\restriction}_{a\,\mathrm{M}}, \vec{\boldsymbol{r}})) \in \Delta$. ⌟

The first restriction in the definition above ensures that any a-edge of $\mathfrak{A}$ only modifies the processes from $a\,\mathrm{M}$ while the other processes are left untouched. The second restriction guarantees that a-edges are independent of the states of the other components in $P \setminus a\,\mathrm{M}$. In this case we can also see Δ as a collection of (local) transition relations Δ_a (for $a \in A$) where $\Delta_a \subseteq \vec{\boldsymbol{Q}}{\restriction}_{a\,\mathrm{M}} \times \vec{\boldsymbol{Q}}{\restriction}_{a\,\mathrm{M}}$. From now on, it suffices to specify the sets Δ_a for any $a \in A$ whenever we want to describe the transitions of such asynchronous automaton.

We can see an asynchronous automaton $\mathfrak{A}$ as a collection of $|P|$ many local NFAs $\mathfrak{A}_i$ (for $i \in P$) with synchronizations in between according to the underlying distributed alphabet $\mathcal{A}$. Consequently, we can also see a run $\vec{\boldsymbol{p}} \xrightarrow{w}_{\mathfrak{A}} \vec{\boldsymbol{q}}$ as a finite collection of local runs $p_i \xrightarrow{\pi_i(w)}_{\mathfrak{A}_i} q_i$ (for $i \in P$). We call such collection an *asynchronous run* of $\mathfrak{A}$ from $\vec{\boldsymbol{p}}$ to $\vec{\boldsymbol{q}}$ labeled with the trace $[w]$. In this case we also write $\vec{\boldsymbol{p}} \xrightarrow{[w]}_{\mathfrak{A}} \vec{\boldsymbol{q}}$. For example, we can visualize an asynchronous run $\vec{\boldsymbol{p}} \xrightarrow{[acba]}_{\mathfrak{A}} \vec{\boldsymbol{q}}$ (if the dependence graph $\mathcal{G}_{\mathcal{A}}$ of $\mathcal{A}$ is $a - b - c$) as in Figure 9.6.

We can find in this figure two possible serializations of this run:

(1) $(p_1, p_2, p_3) \xrightarrow{a}_{\mathfrak{A}} (r_1, r_{2,1}, p_3) \xrightarrow{c}_{\mathfrak{A}} (r_1, r_{2,1}, 2_3) \xrightarrow{b}_{\mathfrak{A}} (r_1, r_{2,2}, q_3) \xrightarrow{a}_{\mathfrak{A}} (q_1, q_2, q_3)$ and

(2) $(p_1, p_2, p_3) \xrightarrow{c}_{\mathfrak{A}} (p_1, p_2, r_3) \xrightarrow{a}_{\mathfrak{A}} (r_1, r_{2,1}, 2_3) \xrightarrow{b}_{\mathfrak{A}} (r_1, r_{2,2}, q_3) \xrightarrow{a}_{\mathfrak{A}} (q_1, q_2, q_3)$.

In other words, the words $acba$ and $caba$ induce the same asynchronous run. Note that this also holds for arbitrary pairs of independent letters:

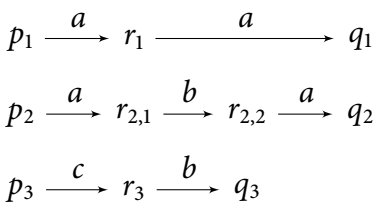

Figure 9.6. Visualization of an asynchronous run Π in an asynchronous automaton.

Observation 9.4.2. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\mathfrak{A} = (\vec{\mathbf{Q}}, A, I, \Delta, F)$ be an asynchronous automaton, $a, b \in A$ with $a \parallel b$, and $\vec{\boldsymbol{p}}, \vec{\boldsymbol{q}}, \vec{\boldsymbol{r}} \in \vec{\mathbf{Q}}$ with $\vec{\boldsymbol{p}} \xrightarrow{a}_{\mathfrak{A}} \vec{\boldsymbol{q}} \xrightarrow{b}_{\mathfrak{A}} \vec{\boldsymbol{r}}$. Set $\vec{\boldsymbol{q}}' := (\vec{\boldsymbol{r}}\restriction_{b\,\mathrm{M}}, \vec{\boldsymbol{p}}\restriction_{P \setminus b\,\mathrm{M}}) \in \vec{\mathbf{Q}}$. Then we have $\vec{\boldsymbol{p}} \xrightarrow{b}_{\mathfrak{A}} \vec{\boldsymbol{q}}' \xrightarrow{a}_{\mathfrak{A}} \vec{\boldsymbol{r}}$.* ◂

By induction on the number of commutations we also learn that if $\vec{\boldsymbol{p}} \xrightarrow{\lambda}_{\mathfrak{A}} \vec{\boldsymbol{q}}$ is an asynchronous run, we find a serialization $\vec{\boldsymbol{p}} \xrightarrow{w}_{\mathfrak{A}} \vec{\boldsymbol{q}}$ for each word $w \in \lambda$. This also implies $L(\mathfrak{A}) = \zeta_{\mathcal{A}}^{-1}(\zeta_{\mathcal{A}}(L(\mathfrak{A})))$. Hence, we infer that the accepted trace language $T(\mathfrak{A})$ of an asynchronous automaton $\mathfrak{A}$ is recognizable. We can also show that each recognizable trace language can be accepted by a (deterministic) asynchronous automaton (cf. [Zie87]).

Now, assume that there is an asynchronous run $\vec{\boldsymbol{p}} \xrightarrow{\lambda}_{\mathfrak{A}} \vec{\boldsymbol{q}}$ of an asynchronous automaton $\mathfrak{A}$ labeled with $\lambda \in \mathbb{M}(\mathcal{A})$ from $\vec{\boldsymbol{p}}$ to $\vec{\boldsymbol{q}}$. We fix one such asynchronous run Π (we also write $\Pi = (\vec{\boldsymbol{p}} \xrightarrow{\lambda}_{\mathfrak{A}} \vec{\boldsymbol{q}})$ in this case). Let $\kappa \in \mathbb{M}(\mathcal{A})$ be a prefix of λ, i.e., $\lambda = \kappa\mu$ for a trace $\mu \in \mathbb{M}(\mathcal{A})$. Then there is a uniquely defined state Π_κ on this run with $\Pi = (\vec{\boldsymbol{p}} \xrightarrow{\kappa}_{\mathfrak{A}} \Pi_\kappa \xrightarrow{\mu}_{\mathfrak{A}} \vec{\boldsymbol{q}})$. Note that Π_κ consists of the states r_i reachable via $\pi_i(\kappa)$ on the local run $p_i \xrightarrow{\pi_i(\lambda)}_{\mathfrak{A}} q_i$. In particular, we have $\Pi_\varepsilon = \vec{\boldsymbol{p}}$ and $\Pi_\lambda = \vec{\boldsymbol{q}}$. For example, in Figure 9.6 we have $\Pi_{[ac]} = (r_1, r_{2,1}, r_3)$.

It is well-known that the class of recognizable trace languages is closed under Boolean operations, concatenation, shuffle, and reversal. Additionally, if $L \subseteq \mathbb{M}(\mathcal{A})$ is recognizable and connected then its iteration L^* is recognizable as well (cf. [DR95]). This statement also holds if we generalize the Kleene closure as follows: let $L \subseteq \mathbb{M}(\mathcal{A})$ be recognizable and connected and let $S \subseteq \mathbb{N}$ be recognizable in $\mathbb{N}$[xv]. Then $L^S = \bigcup_{n \in S} L^n$ is recognizable. Note that we have $L^* = L^{\mathbb{N}}$. All of the closure properties listed above are effective and for a fixed distributed alphabet $\mathcal{A}$ even efficient (cf. Appendix A).

9.5 Distributed Stacks[xvi]

Now, we want to study the reachability problem and the model checking problem of several temporal logics in Kripke-structures induced by special $\mathcal{P}_{\mathcal{A}}$-systems where $\mathcal{A}$ is a distributed alphabet (in the following we call these systems *distributed pushdown systems*). As we have already mentioned in Section 3.4.2 we know that in the general case distributed pushdown

[xv] Note that $S \subseteq \mathbb{N}$ is recognizable in $\mathbb{N}$ iff S is semi-linear, i.e., iff the word language $\{a^n \mid n \in S\}$ is regular.

[xvi] The results in this section stem from an unpublished joint work with Dietrich Kuske [KK22].

systems having at least two stacks are as powerful as Turing-machines. In the following we will recall a proof of this statement:

Lemma 9.5.1. *Let Γ be an alphabet and $L \subseteq \Gamma^*$ be a recursively enumerable language. There is a distributed alphabet $\mathcal{A} = (A, P, \mathrm{M})$ with $\Gamma \subseteq A$ and $|P| = 2$ and a distributed pushdown system $\mathfrak{S}$ over $\mathcal{A}$ such that*

$$L = \{w \in \Gamma^* \mid c \to_{\mathfrak{S}}^* (q, [wx])\}$$

holds for a configuration $c \in \mathrm{Conf}_{\mathfrak{S}}$, a state q of $\mathfrak{S}$, and $x \in (A \setminus \Gamma)^*$.

Proof. Since L is a recursively enumerable language, there is a grammar $\mathfrak{G} = (N, \Gamma, R, S)$ of Chomsky-type 0 with $L(\mathfrak{G}) = L$. We set $\mathcal{A} = (A, P, \mathrm{M})$ as follows:

- $A := N \cup \Gamma \cup N' \cup \Gamma' \cup \{\#, \#'\}$ where N' and Γ' are disjoint copies of N and Γ, respectively, and $\#$ and $\#'$ are new letters - we denote the corresponding copy of any letter $\alpha \in N \cup \Gamma \cup \{\#\}$ by $\alpha' \in N' \cup \Gamma' \cup \{\#'\}$,
- $P := \{1, 2\}$, and
- $\mathrm{M} := \big((N' \cup \Gamma' \cup \{\#'\}) \times \{1\}\big) \cup \big((N \cup \Gamma \cup \{\#\}) \times \{2\}\big)$.

Now, let $k \in \mathbb{N}$ be the maximal length of all words $\ell, r \in \Gamma^*$ with $(\ell, r) \in R$. We construct $\mathfrak{S} = (Q, \mathcal{P}_{\mathcal{A}}, \Delta, \Lambda)$ as follows:

- $Q := \{q_v \mid v \in (N \cup \Gamma)^*, |v| \leq k\}$,
- Λ is arbitrary, and
- Δ consists of the following transitions:
  - (1) $(q_v, \overline{\alpha'}, q_{\alpha v})$ and $(q_v, \overline{\alpha}, q_{v\alpha})$ for any $v \in (N \cup \Gamma)^*$ with $|v| < k$ and $\alpha \in N \cup \Gamma$,
  - (2) $(q_{\alpha v}, \overline{\beta'}\beta'\alpha', q_v)$ and $(q_{v\alpha}, \overline{\beta}\beta\alpha, q_v)$ for any $v \in (N \cup \Gamma)^*$ with $|v| < k$, $\alpha \in N \cup \Gamma$, and $\beta \in N \cup \Gamma \cup \{\#\}$, and
  - (3) $(q_\ell, \overline{\alpha'}\alpha', q_r)$ and $(q_\ell, \overline{\alpha}\alpha, q_r)$ for any production $(\ell, r) \in R$ and $\alpha \in N \cup \Gamma \cup \{\#\}$.

Then for all words $u, w \in (N \cup \Gamma^*)$ and states $q_v \in Q$ we have

$$(q_\varepsilon, [\#' S\#]) \to_{\mathfrak{S}}^* (q_v, [{u'}^{\mathrm{R}}\#' w\#]) \iff S \Rightarrow_{\mathfrak{G}}^* uvw$$

where $u' \in (N' \cup \Gamma')^*$ is the copy of u. In particular, we infer $w \in L$ if, and only if, $(q_\varepsilon, [\#' S\#]) \to_{\mathfrak{S}}^* (q_\varepsilon, [\#' w\#])$ holds for any word $w \in \Gamma^*$. ◂

In other words, the constructed distributed pushdown system is able to copy data from one stack to the other one with the help of the transitions of types (1) and (2). Additionally, the system can do a derivation step of $\mathfrak{G}$ whenever the left-hand side of a rule in $\mathfrak{G}$ is in the bounded memory in the control state. Hence, we can see the contents of the stacks as a prefix and suffix of the non-empty part of a Turing-tape with an infix of bounded length in between (which is stored in the control state). Note that in the constructed system each transition modifies only one stack at once.

Hence, to translate the results from single-pushdown systems to systems having more than one stack as their memory, we have to weaken the expressiveness of these systems. A first approach is to consider systems which are asynchronous in the sense of Definition 9.4.1. This means, for each stack we have an independent "local" control component. Write and read actions of a letter modify exactly the local control components of the processes which are able to handle those letters. Additionally, those actions are independent of the local control components which are unable to handle the associated letter. Unfortunately, such distributed pushdown systems are still Turing-complete:

Remark 9.5.2. Let $\mathfrak{M} = (Q, \mathcal{T}_{B,\square}, \Delta, \Lambda)$ be a Turing-system with blank symbol $\square$ (note that this is a system with a Turing-tape, cf. Section 3.2.6). We can assume that any transition is labeled by $\overline{B}B\{L, R, N\}$. Then we can construct a distributed pushdown system $\mathfrak{S}$ as follows: we have two stacks where stack 1 contains the Turing-tape's (non-empty) contents, which are left of the head position, and stack 2 contains the remaining (non-empty) contents such that the head position is the top of stack 2. To this end, we have two copies a_1 and a_2 of each letter $a \in B$ where a_i can be handled by stack i. Then the transitions of our system simulate the Turing-tape's transformations. Since we simulate our transitions only on stack 2, the head-movements to the left-hand side are a bit more tricky since we cannot move stack 1's top element in just one step. So, we do this as follows: if stack 2 is in state q it moves into a copy q' of this state indicating that this stack waits for a letter from stack 1. Then the first stack reads its top element c_1 and pushes a marked copy c_2' to stack 2 (note that c_2' indicates that it is new on this stack). Finally, stack 2 moves from the waiting state q' to q and converts the marked copy c_2' into c_2.

All in all, we construct the following distributed alphabet $\mathcal{A} = (A, P, \mathrm{M})$:

- $A := \{a_1, a_2, a_2' \mid a \in B\} \cup \{\#_1, \#_2\}$ where a_1, a_2, a_2' are three distinct copies of a,
- $P := \{1, 2\}$, and
- $\mathrm{M} := \{(a_1, 1), (a_2, 2), (a_2', 2) \mid a \in B\} \cup \{(\#_1, 1), (\#_2, 2)\}$.

The described distributed pushdown system $\mathfrak{S} = (\vec{\boldsymbol{Q}}, \mathcal{P}_{\mathcal{A}}, \Delta', \Lambda')$ is the following:

- $\vec{\boldsymbol{Q}} := Q_1 \times Q_2$ where $Q_1 := \{\top\}$ and $Q_2 := \{q, q' \mid q \in Q\}$ where q' is a distinct copy of q,
- $\Lambda'(\top, q) = \Lambda'(\top, q') := \Lambda(q)$ where $q \in Q$, and
- Δ' consists of the following transitions:
  (1) $((\top, p), \overline{a_2}b_2, (\top, q))$ for each $(p, \overline{a}bN, q) \in \Delta$,
  (2) $((\top, p), \overline{a_2}b_1, (\top, q))$ for each $(p, \overline{a}bR, q) \in \Delta$,
  (3) $((\top, p), \overline{a_2}b_2, (\top, q'))$, $((\top, s), \overline{c_1}c_2', (\top, s))$, and $((\top, q'), \overline{c_2'}c_2, (\top, q))$ for each $(p, \overline{a}bL, q) \in \Delta$, $c \in B$, and $s \in Q_2$, and
  (4) $((\top, s), \overline{\#_i}\#_i\square_i, (\top, s))$ for each $i \in \{1, 2\}$ and $s \in Q_2$.

Then we have the following equivalence:

$$pu \to_{\mathfrak{M}}^* vqw \iff \exists i, j \in \mathbb{N}\colon ((\top, p), [\#_1\, u_2\#_2]) \to_{\mathfrak{S}}^* ((\top, q), [{v_1}^{\mathrm{R}}\square_1^i\#_1\, w_2\square_2^j\#_2])$$

for each $u, v, w \in B^*$ and $p, q \in Q$. Note that w_i means the i^{th} copy of w for each $i \in \{1, 2\}$. Hence, if we assume $\iota, f \in Q$ to be an initial resp. final state of $\mathfrak{M}$ the language

$$\{w \in B^* \mid \exists i, j \in \mathbb{N}\colon ((\top, \iota), [\#_1\, w_2 \#_2]) \to_{\mathfrak{S}}^* ((\top, f), [\square_1^i \#_1\, \square_2^j \#_2])\}$$

is exactly the set of words the Turing-machine obtained from $\mathfrak{M}$ with initial state ι and final state f would accept. ⌟

In Lemma 9.5.1 we have seen that distributed pushdown systems are as powerful as Turing-machines if each transition writes letters at most into those stacks from which we have read a letter right before. Additionally, Remark 9.5.2 shows that distributed pushdown systems are Turing-complete if their control components are composed of local components for each process. Note that in this case we have transitions writing letters into stacks from which we have not read a letter before (these are the transitions $((\top, p), \overline{a_2} b_1, (\top, q))$ and $((\top, s), \overline{c_1} c_2', (\top, s))$ in the constructed distributed pushdown system). Now, we want to consider the combination of both restrictions to distributed pushdown systems. This means, each process has its own local control component and we write only letters into those stacks from which we have read another one right before. These special systems are given in the following definition. Afterwards, we will consider the reachability problems of such systems.

Definition 9.5.3. Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet. An *asynchronous pushdown system* is a distributed pushdown system $\mathfrak{S} = (\vec{\boldsymbol{Q}}, \mathcal{P}_{\mathcal{A}}, \Delta, \Lambda)$ where $\vec{\boldsymbol{Q}} = \prod_{i \in P} Q_i$ is a finite product of finite sets Q_i (for any $i \in P$) and where each transition $(\vec{\boldsymbol{p}}, t, \vec{\boldsymbol{q}}) \in \Delta$ satisfies the following properties:

(1) $t = \overline{a} s$ with $a \in A$ and $s \in A^*$ such that $s\,\mathrm{M} \subseteq a\,\mathrm{M}$,

(2) $\vec{\boldsymbol{p}}{\restriction}_{P \setminus a\,\mathrm{M}} = \vec{\boldsymbol{q}}{\restriction}_{P \setminus a\,\mathrm{M}}$, and

(3) for each $\vec{\boldsymbol{r}} \in \vec{\boldsymbol{Q}}{\restriction}_{P \setminus a\,\mathrm{M}}$ we have $((\vec{\boldsymbol{p}}{\restriction}_{a\,\mathrm{M}}, \vec{\boldsymbol{r}}), t, (\vec{\boldsymbol{q}}{\restriction}_{a\,\mathrm{M}}, \vec{\boldsymbol{r}})) \in \Delta$. ⌟

We can also define *asynchronous pushdown automata* with the same restrictions.

The first condition ensures that we write letters only into those stacks from which we have read a letter before. The second and third condition in this definition ensure that a transition $(\vec{\boldsymbol{p}}, \overline{a} t, \vec{\boldsymbol{q}}) \in \Delta$ of $\mathfrak{S}$ is independent of the processes in $P \setminus a\,\mathrm{M}$. Hence, we can see Δ as the union of finitely many finite sets

$$\Delta_a \subseteq \vec{\boldsymbol{Q}}{\restriction}_{a\,\mathrm{M}} \times A^* \times \vec{\boldsymbol{Q}}{\restriction}_{a\,\mathrm{M}}$$

where $a \in A$. From now on, we may also define Δ with the help of these sets Δ_a.

Recall the constructed systems from the proof of Lemma 9.5.1 and in Remark 9.5.2. The system in Lemma 9.5.1 only satisfies the first condition in Definition 9.5.3. The system in Remark 9.5.2 violates this condition, but satisfies the second and third one. Hence, none of these two constructed systems is asynchronous in the sense of this definition.

To simplify notations, we will introduce a quasi-ordering $\leq$ on the alphabet A. Concretely, for $a, b \in A$ we write $a \leq b$ if, and only if, $a\,\mathrm{M} \subseteq b\,\mathrm{M}$. From this quasi-ordering we also obtain the following subalphabets of A for any letter $a \in A$:

$$A_{\leq a} := \{b \in A \mid b \leq a\} \quad \text{and} \quad A_{=a} := \{b \in A \mid a \leq b, b \leq a\}.$$

Hence, the alphabets $A_{=a}$ are exactly the equivalence classes induced by $\leq$ and, therefore, form a partition of A. The alphabets $A_{\leq a}$ also have a natural meaning in terms of asynchronous

pushdown systems: whenever we read a letter $a \in A$ from the stack's top, we can only write letters at most on the processes from $a\,\mathrm{M}$. In other words, for a transition $(\vec{p}, \overline{a}t, \vec{q})$ of an asynchronous pushdown system we know that $t \in A^*_{\leq a}$ holds. We can also extend the quasi-ordering $\leq$ to words on A. So, we write $v \leq w$ if, and only if, $v\,\mathrm{M} \subseteq w\,\mathrm{M}$ holds. Consequently, we also have $t \leq a$ for any transition $(\vec{p}, \overline{a}t, \vec{q})$ of an asynchronous pushdown system.

Example 9.5.4. Let $\mathcal{A} = (\{a, b, c\}, \{1, 2\}, \{(a, 1), (a, 2), (b, 1), (c, 2)\})$ be a distributed alphabet. Then we have $b, c \leq a$. Now, consider the distributed pushdown system $\mathfrak{S}$ from Figure 9.7. The set of control states of $\mathfrak{S}$ is the product $\{0, 1\} \times \{0, 1\}$. Additionally, $\overline{b}$-edges depend only on process 1 and $\overline{c}$-edges depend on process 2. Since $b, c \leq a$ holds, we finally see that $\mathfrak{S}$ is asynchronous. ⌟

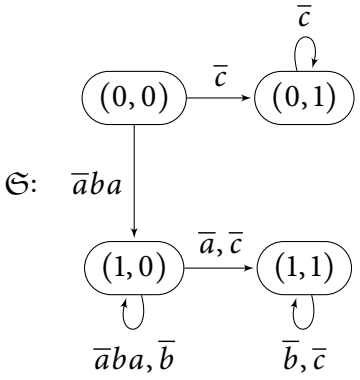

Figure 9.7. The asynchronous pushdown system $\mathfrak{S}$.

Now, let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet. If P is a singleton, then any distributed pushdown system over $\mathcal{A}$ is obviously asynchronous. In this case, a distributed pushdown system is a reliable (single-)pushdown system as considered in Section 9.2.

Next, let P be an arbitrary set of processes. Consider an asynchronous pushdown system $\mathfrak{S}$ over $\mathcal{A}$ with a configuration $(\vec{p}, [v]) \in \mathrm{Conf}_{\mathfrak{S}}$. If $\mathfrak{S}$ is in this configuration, we can only read the top letters from the stacks in $v\,\mathrm{M}$ and write letters to at most these stacks. Additionally, the local states of the processes in $P \setminus v\,\mathrm{M}$ remain untouched. After multiple transitions we have still touched at most the stacks from $v\,\mathrm{M}$. This is formalized in the following observation:

Observation 9.5.5. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\mathfrak{S} = (\vec{Q}, \mathcal{P}_{\mathcal{A}}, \Delta, \Lambda)$ be an asynchronous pushdown system, and $(\vec{p}, [v]), (\vec{q}, [w]) \in \mathrm{Conf}_{\mathfrak{S}}$ with $(\vec{p}, [v]) \to^*_{\mathfrak{S}} (\vec{q}, [w])$. Then we have*

(1) $w \leq v$ and

(2) $\vec{p}\restriction_{P \setminus v\,\mathrm{M}} = \vec{q}\restriction_{P \setminus v\,\mathrm{M}}$.

Proof. We prove this by induction on the length $k \in \mathbb{N}$ of the path $(\vec{p}, [v]) \to^k_{\mathfrak{S}} (\vec{q}, [w])$. The case $k = 0$ is obvious since $(\vec{p}, [v]) = (\vec{q}, [w])$ holds.

Now, let $k \geq 0$. From $(\vec{p}, [v]) \to_{\mathfrak{S}}^{k+1} (\vec{q}, [w])$ we infer the existence of $(\vec{r}, [u]) \in \mathrm{Conf}_{\mathfrak{S}}$ with

$$(\vec{p}, [v]) \to_{\mathfrak{S}}^{k} (\vec{r}, [u]) \to_{\mathfrak{S}} (\vec{q}, [w]) .$$

By induction hypothesis we have $u \leq v$ and $\vec{p}\restriction_{P \setminus v\,\mathrm{M}} = \vec{r}\restriction_{P \setminus v\,\mathrm{M}}$. Additionally, by definition of $\to_{\mathfrak{S}}$ there are $a \in A$ and $t, x \in A^*$ with $(\vec{r}, \overline{a}t, \vec{q}) \in \Delta$, $u \approx_{\mathcal{A}} ax$, and $w \approx_{\mathcal{A}} t^{\mathrm{R}}x$ (i.e., $[\![\overline{a}t]\!]([u]) = [w]$). Since $\mathfrak{S}$ is asynchronous, we have $t \leq a$ implying

$$w\,\mathrm{M} = t\,\mathrm{M} \cup x\,\mathrm{M} \subseteq a\,\mathrm{M} \cup x\,\mathrm{M} = u\,\mathrm{M} \subseteq v\,\mathrm{M} ,$$

i.e., $w \leq v$. Additionally, from the asynchronism of $\mathfrak{S}$ we have $\vec{r}\restriction_{P \setminus a\,\mathrm{M}} = \vec{q}\restriction_{P \setminus a\,\mathrm{M}}$. Due to $a\,\mathrm{M} \subseteq u\,\mathrm{M} \subseteq v\,\mathrm{M}$ we finally infer

$$\vec{p}\restriction_{P \setminus v\,\mathrm{M}} = \vec{r}\restriction_{P \setminus v\,\mathrm{M}} = \vec{q}\restriction_{P \setminus v\,\mathrm{M}} . \qquad \blacktriangleleft$$

Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\mathfrak{S} = (\vec{\mathbf{Q}}, \mathcal{P}_{\mathcal{A}}, \Delta, \Lambda)$ be an asynchronous pushdown system. For an edge $(\vec{p}, \kappa) \to_{\mathfrak{S}} (\vec{q}, \lambda)$ in the configuration graph $\mathcal{M}(\mathfrak{S})$ we can also specify the action sequence $t \in \Sigma^*$ we have executed in the corresponding transition. Concretely, we write $(\vec{p}, \kappa) \xrightarrow{t}_{\mathfrak{S}} (\vec{q}, \lambda)$ if we have $(\vec{p}, t, \vec{q}) \in \Delta$ with $[\![t]\!](\kappa) = \lambda$. We can also extend this notion to arbitrary sequences of actions. In other words, we write $(\vec{p}, \kappa) \xrightarrow{s}_{\mathfrak{S}} (\vec{q}, \lambda)$ if there is an s-labeled path from $(\vec{p}, \kappa)$ to $(\vec{q}, \lambda)$ in the configuration graph.

Recall that in asynchronous automata we are able to commute independent letters according to Observation 9.4.2. We are in a similar situation in asynchronous pushdown systems: for example, consider the transitions $((1,0), \overline{b}, (1,0))$ and $((1,0), \overline{c}, (1,1))$ of the asynchronous pushdown system $\mathfrak{S}$ in Figure 9.7. Note that we have $b \parallel c$ in this case. Since the former transition only modifies process 1 and the latter one modifies only process 2, it does not matter whether we first read b and then c or vice versa. In both cases, we end up in the same configuration. This is depicted in Figure 9.8.

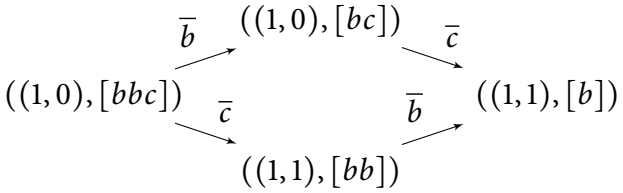

Figure 9.8. Two possible computations of the asynchronous pushdown system $\mathfrak{S}$ in Figure 9.7.

This fact can also be generalized to arbitrary transition sequences $s, t \in \Sigma^*$ in any asynchronous pushdown system:

Observation 9.5.6. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\mathfrak{S}$ be an asynchronous pushdown system, $s, t \in \Sigma^*$, and $c, d, e \in \mathrm{Conf}_{\mathfrak{S}}$ with $s \parallel t$ and $c \xrightarrow{s}_{\mathfrak{S}} d \xrightarrow{t}_{\mathfrak{S}} e$. Then there is $d' \in \mathrm{Conf}_{\mathfrak{S}}$ with $c \xrightarrow{t}_{\mathfrak{S}} d' \xrightarrow{s}_{\mathfrak{S}} e$.* $\blacktriangleleft$

Next, we need an automaton model which accepts sets of configurations of distributed pushdown systems. For non-distributed pushdown systems $\mathfrak{S}$ we used NFAs $\mathfrak{A}$ having the states from $\mathfrak{S}$ as their initial states and we have $(q, w) \in C(\mathfrak{A})$ if there is an accepting run in $\mathfrak{A}$ starting in q and labeled with w. Since $C(\mathfrak{A})$ can be seen as a finite collection of regular languages we also called $C(\mathfrak{A})$ regular.

Now, let $\mathfrak{S}$ be a distributed pushdown system. Then we represent the distributed stack's contents by traces. Recall that the class of rational trace languages does not coincide with the class of recognizable ones in most trace monoids. For example, the trace language $(ab)^*$ is rational but not recognizable if $a, b \in A$ are two independent letters (i.e., if we have $a \parallel b$). Those two classes are accepted by two different automaton models. So, the rational trace languages are accepted by NFAs while the recognizable ones are accepted by restricted NFAs - the so-called asynchronous automata. Accordingly, to translate $\mathfrak{S}$-NFAs to the distributed case we also have two automaton models with different expressibility:

Definition 9.5.7. Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\mathfrak{S} = (\vec{\boldsymbol{Q}}, \mathcal{P}_{\mathcal{A}}, \Delta, \Lambda)$ be an asynchronous pushdown system. An $\mathfrak{S}$-*NFA* is an NFA $\mathfrak{A} = (S, A, \vec{\boldsymbol{Q}}, \delta, F)$ with $\vec{\boldsymbol{Q}} \subseteq S$. $\mathfrak{A}$ is called an $\mathfrak{S}$-*asynchronous automaton* if it is asynchronous.

The $\mathfrak{S}$-NFA $\mathfrak{A}$ *accepts* the following set of configurations of $\mathfrak{S}$

$$C(\mathfrak{A}) := \{(\vec{\boldsymbol{q}}, \lambda) \mid \vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}, \vec{\boldsymbol{q}} \xrightarrow{\lambda}_{\mathfrak{A}} F\}.$$

A set of configurations $C \subseteq \mathrm{Conf}_{\mathfrak{S}}$ is called *rational* if there is an $\mathfrak{S}$-NFA $\mathfrak{A}$ with $C(\mathfrak{A}) = C$. It is called *recognizable* if there is an $\mathfrak{S}$-asynchronous automaton with $C(\mathfrak{A}) = C$. ⌟

We can see the transition relation δ of an $\mathfrak{S}$-asynchronous automaton $\mathfrak{A} = (\vec{\boldsymbol{S}}, \mathcal{A}, \vec{\boldsymbol{Q}}, \delta, F)$ as a finite collection of sets $\delta_a \subseteq \vec{\boldsymbol{S}}{\restriction_{a\,\mathrm{M}}} \times \vec{\boldsymbol{S}}{\restriction_{a\,\mathrm{M}}}$ where $a \in A$ is a letter.

Note that the notion of an $\mathfrak{S}$-NFA is similar to the one-stack case. The only difference is the underlying system which is now asynchronous and the accepted configurations which contain traces instead of words. Again, we can understand $C(\mathfrak{A})$ of an $\mathfrak{S}$-NFA $\mathfrak{A}$ as a collection of a finite number of rational trace languages. Similarly, if $\mathfrak{A}$ is asynchronous, $C(\mathfrak{A})$ is a finite collection of recognizable trace languages. This is formalized in the following fact:

Fact 9.5.8. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\mathfrak{S} = (\vec{\boldsymbol{Q}}, \mathcal{P}_A, \Delta, \Lambda)$ be an asynchronous pushdown system. Then the following statements hold:*

(1) Let $\mathfrak{A}$ be an $\mathfrak{S}$-NFA and $\vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}$. Then $\vec{\boldsymbol{q}}\,C(\mathfrak{A})$ is rational in $\mathbb{M}(\mathcal{A})$. If $\mathfrak{A}$ is asynchronous then $\vec{\boldsymbol{q}}\,C(\mathfrak{A})$ is even recognizable.

(2) Let $L \subseteq \mathbb{M}(\mathcal{A})$ be rational. Then there is an $\mathfrak{S}$-NFA $\mathfrak{A}$ and $\vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}$ with $L = \vec{\boldsymbol{q}}\,C(\mathfrak{A})$.

(3) Let $L \subseteq \mathbb{M}(\mathcal{A})$ be recognizable. Then there is an $\mathfrak{S}$-asynchronous automaton $\mathfrak{A}$ and $\vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}$ with $L = \vec{\boldsymbol{q}}\,C(\mathfrak{A})$. ◂

Now, we want to consider the sets of forwards and backwards reachable configurations of a given asynchronous pushdown system. In contrast to the non-distributed case, we have to consider backwards reachability and forwards reachability separately. Concretely, we

will see that the map $\mathrm{pre}^*_{\mathfrak{S}} \colon 2^{\mathrm{Conf}_{\mathfrak{S}}} \to 2^{\mathrm{Conf}_{\mathfrak{S}}}$ preserves recognizability effectively and even efficiently. We will see, that our construction is a generalization of the already mentioned construction from [BEM97], where we computed $\mathrm{pre}^*_{\mathfrak{S}}$ in the non-distributed case. On the other hand, we will show that forwards reachability does not preserve recognizability. Concretely, for any rational trace language $L \subseteq \mathbb{M}(\mathcal{A})$ we will find an asynchronous pushdown system $\mathfrak{S}$, a configuration $c \in \mathrm{Conf}_{\mathfrak{S}}$, and a control state $\vec{q}$ of $\mathfrak{S}$ such that L is the set of all configurations with control state $\vec{q}$ which are (forwards) reachable from c (i.e., we have $L = \vec{q}\,\mathrm{post}^*_{\mathfrak{S}}(c)$). However, we can prove that for any asynchronous pushdown system $\mathfrak{S}$ the map $\mathrm{post}^*_{\mathfrak{S}} \colon 2^{\mathrm{Conf}_{\mathfrak{S}}} \to 2^{\mathrm{Conf}_{\mathfrak{S}}}$ preserves rationality effectively.

9.5.1 Backwards Reachability

First, we consider backwards reachability in asynchronous pushdown automata. We will prove now that pre* efficiently preserves recognizability:

Theorem 9.5.9. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\mathfrak{S}$ be an asynchronous $\mathcal{P}_{\mathcal{A}}$-system, and $C \subseteq \mathrm{Conf}_{\mathfrak{S}}$ be recognizable. Then $\mathrm{pre}^*_{\mathfrak{S}}(C)$ is recognizable. We can compute an $\mathfrak{S}$-asynchronous automaton accepting $\mathrm{pre}^*_{\mathfrak{S}}(C)$ from $\mathfrak{S}$ and an $\mathfrak{S}$-asynchronous automaton accepting C in polynomial time.*

We generalize the construction from [BEM97] to $\mathfrak{S}$-asynchronous automata. So, let $\mathfrak{S} = (\vec{\boldsymbol{Q}}, \mathcal{P}_{\mathcal{A}}, \Delta, \Lambda)$ and $\mathfrak{A}^{(0)} = (\vec{\boldsymbol{S}}, A, \vec{\boldsymbol{Q}}, \delta^{(0)}, F)$ be an $\mathfrak{S}$-asynchronous automaton accepting C. We can assume that $\delta^{(0)}_a \subseteq \vec{\boldsymbol{S}}{\restriction_{a\,\mathrm{M}}} \times \vec{\boldsymbol{T}}{\restriction_{a\,\mathrm{M}}}$ holds for each $a \in A$ where $\vec{\boldsymbol{T}} := \prod_{i \in P} S_i \setminus Q_i$. In other words, no local state $q_i \in Q_i$ has an in-edge in $\mathfrak{A}^{(0)}$. This is no restriction since we can construct such restricted $\mathfrak{S}$-asynchronous automaton in polynomial time: for each local state $q_i \in Q_i$ (where $i \in P$) we can add a copy q_i' to $S_i \setminus Q_i$. Then q_i' has the same out-edges as q_i and each in-edge of q_i is replaced by one pointing to q_i' instead.

We inductively compute $\mathfrak{S}$-asynchronous automata $\mathfrak{A}^{(k)} = (\vec{\boldsymbol{S}}, \mathcal{A}, \vec{\boldsymbol{Q}}, \delta^{(k)}, F)$ as follows: for $k \in \mathbb{N}$ and $a \in A$ we set

$$\delta^{(k+1)}_a := \delta^{(k)}_a \cup \left\{ (\vec{\boldsymbol{p}}{\restriction_{a\,\mathrm{M}}}, \vec{\boldsymbol{s}}{\restriction_{a\,\mathrm{M}}}) \;\middle|\; \vec{\boldsymbol{p}} \in \vec{\boldsymbol{Q}}, \vec{\boldsymbol{s}} \in \vec{\boldsymbol{S}}, \exists \vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}} \colon (\vec{\boldsymbol{p}}{\restriction_{a\,\mathrm{M}}}, t, \vec{\boldsymbol{q}}{\restriction_{a\,\mathrm{M}}}) \in \Delta_a \;\&\; \vec{\boldsymbol{q}} \xrightarrow{[t^{\mathrm{R}}]}_{\mathfrak{A}^{(k)}} \vec{\boldsymbol{s}} \right\}.$$

We can understand this construction as follows: whenever we have a transition $(\vec{\boldsymbol{p}}, \overline{a}t, \vec{\boldsymbol{q}})$ in $\mathfrak{S}$ and a configuration $(\vec{\boldsymbol{q}}, [t^{\mathrm{R}}w]) \in \mathrm{pre}^*_{\mathfrak{S}}(C)$ we also have $(\vec{\boldsymbol{p}}, [aw]) \in \mathrm{pre}^*_{\mathfrak{S}}(C)$ since $[\![\overline{a}t]\!]([aw]) = [t^{\mathrm{R}}w]$ holds. This construction is depicted in Figure 9.9. Note that the transition $(\vec{\boldsymbol{p}}, \overline{a}t, \vec{\boldsymbol{q}}) \in \Delta$ as well as the run $\vec{\boldsymbol{q}} \xrightarrow{[t^{\mathrm{R}}]}_{\mathfrak{A}^{(k)}} \vec{\boldsymbol{s}}$ operates on components from $a\,\mathrm{M} \supseteq t\,\mathrm{M}$, only. Additionally, both do not depend on the components from $P \setminus a\,\mathrm{M}$. Therefore, the same applies to the new transition $(\vec{\boldsymbol{p}}, a, \vec{\boldsymbol{s}}) \in \delta^{(k+1)}$ ensuring that $\mathfrak{A}^{(k+1)}$ also is asynchronous. We should also note that this argument requires $t \leq a$ and would therefore not work for arbitrary distributed pushdown systems.

Let $\mathfrak{A}^{(\infty)} = (\vec{\boldsymbol{S}}, \mathcal{P}_{\mathcal{A}}, \vec{\boldsymbol{Q}}, \delta^{(\infty)}, F)$ be the "limit" of the sequence $(\mathfrak{A}^{(k)})_{k \in \mathbb{N}}$ satisfying $\delta^{(\infty)} = \bigcup_{k \in \mathbb{N}} \delta^{(k)}$. Since all $\mathfrak{A}^{(k)}$ are asynchronous, $\mathfrak{A}^{(\infty)}$ also is asynchronous. Our next aim is, to prove that $C(\mathfrak{A}^{(\infty)}) = \mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{A}^{(0)}))$ holds. Afterwards we will also show that $\mathfrak{A}^{(\infty)}$ can also be constructed from the $\mathfrak{S}$-asynchronous automaton $\mathfrak{A}^{(0)}$ in polynomial time. But before, we want to visualize the algorithm with the help of the following example:

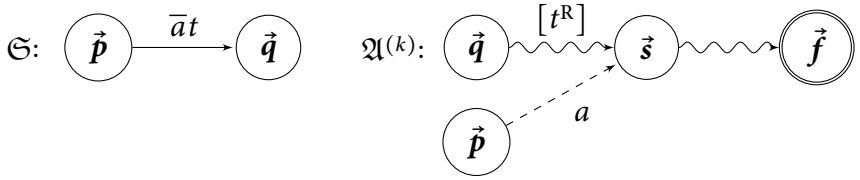

Figure 9.9. Visualization of the construction of $\delta^{(k+1)}$. We add the dashed a-edge in step $k+1$.

Example 9.5.10. Recall the asynchronous pushdown system $\mathfrak{S}$ from Example 9.5.4. In Figure 9.10 we depict our algorithm on input $\mathfrak{S}$ and the configuration $((1,1),\varepsilon)$. We obtain an $\mathfrak{S}$-asynchronous automaton $\mathfrak{A}^{(\infty)}$ accepting

$$\big(\{(0,0)\} \times a\{b,c\}^*\big) \cup \big(\{(1,0)\} \times b^*\{a,c\}\{b,c\}^*\big) \cup \big(\{(1,1)\} \times \{b,c\}^*\big)$$

which is exactly $\mathrm{pre}^*_{\mathfrak{S}}(\{((1,1),\varepsilon)\})$.

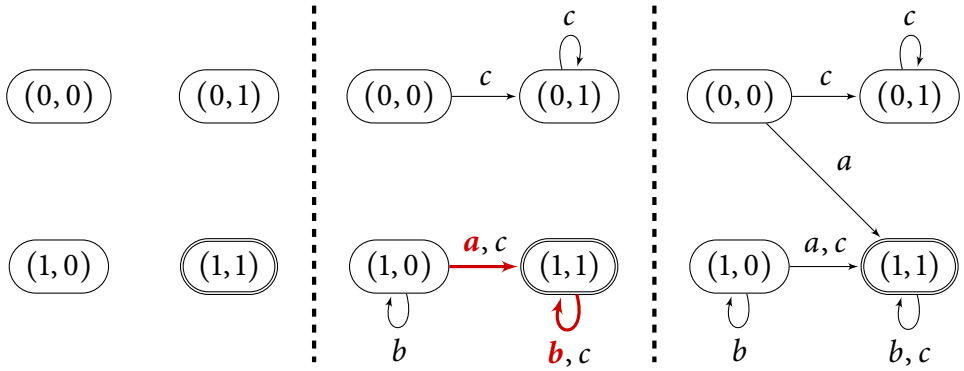

Figure 9.10. The $\mathfrak{S}$-asynchronous automata $\mathfrak{A}^{(0)}$, $\mathfrak{A}^{(1)}$, and $\mathfrak{A}^{(2)}$ (from left to right). We can show that $\mathfrak{A}^{(2)} = \mathfrak{A}^{(3)} = \mathfrak{A}^{(4)} = \dots$ holds. Hence, we have $\mathfrak{A}^{(\infty)} = \mathfrak{A}^{(2)}$.
In the first step we only copy edges from $\mathfrak{S}$ which do not write any letter, since $\mathfrak{A}^{(0)}$ has no edges (i.e., the stacks are empty at this point). In the second step we are able to add the a-edge from $(0,0)$ to $(1,1)$ due to the thick, red-colored path and the transition $((0,0),\overline{a}ba,(1,0))$. Note that every node in this figure is initial - we omitted the corresponding arrows for better readability.

Now, we show $C(\mathfrak{A}^{(\infty)}) = \mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{A}^{(0)}))$ with the help of the following two lemmas each stating one inclusion. Note that the proofs of these lemmas are close to (but a bit more involved than) the ones known from [BEM97].

Lemma 9.5.11. *Let $k \in \mathbb{N}$. Then we have $\mathrm{pre}^k_{\mathfrak{S}}(C(\mathfrak{A}^{(0)})) \subseteq C(\mathfrak{A}^{(k)})$. In particular, we have $\mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{A}^{(0)})) \subseteq C(\mathfrak{A}^{(\infty)})$.*

Proof. We prove the first statement by induction on $k \in \mathbb{N}$. The case $k = 0$ is obvious by definition of $\mathrm{pre}^0_{\mathfrak{S}}$. Now, let $k \geq 0$ and $(\vec{q},[v]) \in \mathrm{pre}^{k+1}_{\mathfrak{S}}(C(\mathfrak{A}^{(0)}))$. Then there is $(\vec{p},[u]) \in \mathrm{pre}^k_{\mathfrak{S}}(C(\mathfrak{A}^{(0)}))$ with $(\vec{q},[v]) \to_{\mathfrak{S}} (\vec{p},[u])$. There exists $(\vec{p},\overline{a}t,\vec{q}) \in \Delta$ and $x \in A^*$ with

$v \approx_{\mathcal{A}} ax$ and $u \approx_{\mathcal{A}} t^{\mathrm{R}}x$ (i.e., we have $[\![\overline{a}t]\!]([v]) = [u]$). By the induction hypothesis we have $(\vec{p}, [t^{\mathrm{R}}x]) = (\vec{p}, [u]) \in C(\mathfrak{A}^{(k)})$. Hence, there are $\vec{s} \in \vec{S}$ and $\vec{f} \in F$ with

$$\vec{p} \xrightarrow{[t^{\mathrm{R}}]}_{\mathfrak{A}^{(k)}} \vec{s} \xrightarrow{[x]}_{\mathfrak{A}^{(k)}} \vec{f}\,.$$

By $(\vec{p}, \overline{a}t, \vec{q}) \in \Delta$ and $\vec{p} \xrightarrow{[t^{\mathrm{R}}]}_{\mathfrak{A}^{(k)}} \vec{s}$ we obtain $(\vec{q}, a, \vec{s}) \in \delta^{(k+1)}$ and, hence,

$$\vec{q} \xrightarrow{[a]}_{\mathfrak{A}^{(k+1)}} \vec{s} \xrightarrow{[x]}_{\mathfrak{A}^{(k)}} \vec{f}\,.$$

Since $\delta^{(k)} \subseteq \delta^{(k+1)}$ we finally obtain $(\vec{q}, [ax]) = (\vec{q}, [v]) \in C(\mathfrak{A}^{(k+1)})$.

Towards the second statement recall that we have $\delta^{(0)} \subseteq \delta^{(1)} \subseteq \delta^{(2)} \subseteq \cdots \subseteq \delta^{(\infty)}$ implying

$$C(\mathfrak{A}^{(0)}) \subseteq C(\mathfrak{A}^{(1)}) \subseteq C(\mathfrak{A}^{(2)}) \subseteq \cdots \subseteq C(\mathfrak{A}^{(\infty)})\,.$$

Then we have

$$\mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{A}^{(0)})) = \bigcup_{k\in\mathbb{N}} \mathrm{pre}^k_{\mathfrak{S}}(C(\mathfrak{A}^{(0)})) \subseteq \bigcup_{k\in\mathbb{N}} C(\mathfrak{A}^{(k)}) \subseteq C(\mathfrak{A}^{(\infty)})\,. \qquad \blacktriangleleft$$

Lemma 9.5.12. *We have* $C(\mathfrak{A}^{(\infty)}) \subseteq \mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{A}^{(0)}))$.

Proof. We prove this lemma with the help of the following statement:

▷ *Claim.* *Let $k \in \mathbb{N}$, $w \in A^*$, $\vec{q} \in \vec{Q}$, and $\vec{s} \in \vec{S}$ with $\vec{q} \xrightarrow{[w]}_{\mathfrak{A}^{(k)}} \vec{s}$. There are a state $\vec{p} \in \vec{Q}$ and a word $v \in A^*$ with the following properties:*

(1) $(\vec{q}, [w]) \to^*_{\mathfrak{S}} (\vec{p}, [v])$ *and*

(2) $\vec{p} \xrightarrow{[v]}_{\mathfrak{A}^{(0)}} \vec{s}$.

Proof. From $\vec{q} \xrightarrow{[w]}_{\mathfrak{A}^{(k)}} \vec{s}$ we infer the existence of $a_1, a_2, \ldots, a_n \in A$ and $\vec{s_0}, \vec{s_1}, \ldots, \vec{s_n} \in \vec{S}$ with

$$\vec{q} = \vec{s_0} \xrightarrow{a_1}_{\mathfrak{A}^{(k)}} \vec{s_1} \xrightarrow{a_2}_{\mathfrak{A}^{(k)}} \cdots \xrightarrow{a_n}_{\mathfrak{A}^{(k)}} \vec{s_n} = \vec{s}$$

and $w = a_1 a_2 \ldots a_n$. We prove the properties (1) and (2) by induction on $k \in \mathbb{N}$ and the size of

$$I := \{1 \le j \le n \mid (\vec{s_{j-1}}, a_j, \vec{s_j}) \in \delta^{(k)} \setminus \delta^{(k-1)}\}\,.$$

Let $k = 0$. Then we obtain our claim with $(\vec{p}, [v]) := (\vec{q}, [w])$. So, from now on assume $k \ge 1$. If $I = \varnothing$ we have $\vec{q} \xrightarrow{[w]}_{\mathfrak{A}^{(k-1)}} \vec{s}$. Then by application of the induction hypothesis we are done. We now assume $|I| \ge 1$. Then there are $w_1, w_2 \in A^*$, $a \in A$, and $\vec{p}, \vec{r} \in \vec{S}$ with

$$w = w_1 a w_2\,, \tag{9.1}$$

$(\vec{p}, a, \vec{r}) \in \delta^{(k)} \setminus \delta^{(k-1)}$, and $\vec{q} \xrightarrow{[w_1]}_{\mathfrak{A}^{(k)}} \vec{p} \xrightarrow{a}_{\mathfrak{A}^{(k)}} \vec{r} \xrightarrow{[w_2]}_{\mathfrak{A}^{(k-1)}} \vec{s}$. By induction hypothesis there are $\vec{q}' \in \vec{Q}$ and $v \in A^*$ with

$$(\vec{q}, [w_1]) \to^*_{\mathfrak{S}} (\vec{q}', [v]) \tag{9.2}$$

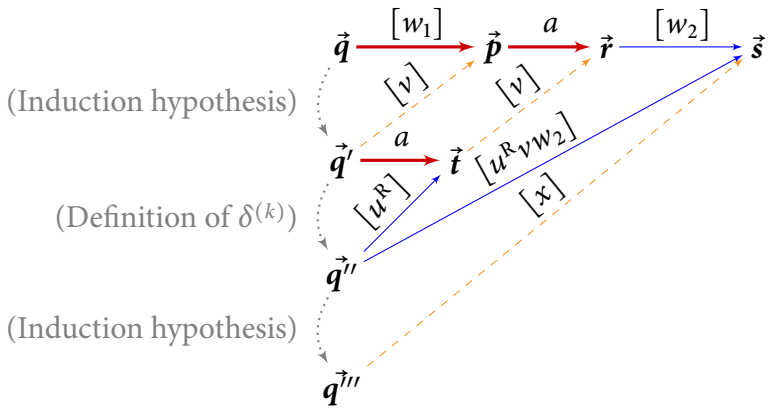

Figure 9.11. Visualization of the induction step of the claim's proof. Here, red-colored (thick) edges represent paths in $\mathfrak{A}^{(k)}$, blue (thin) ones represent paths in $\mathfrak{A}^{(k-1)}$, and orange (dashed) ones represent paths in $\mathfrak{A}^{(0)}$.

and $\vec{q}' \xrightarrow{[v]}_{\mathfrak{A}^{(0)}} \vec{p}$.

Additionally, by $(\vec{p}, a, \vec{r}) \in \delta^{(k)} \setminus \delta^{(k-1)}$ and the definition of $\delta^{(k)}$ we know $\vec{p}\restriction_{a\,\mathrm{M}} \in \vec{Q}\restriction_{a\,\mathrm{M}}$. Since $\delta_a^{(0)} \subseteq \vec{S}\restriction_{a\,\mathrm{M}} \times \vec{T}\restriction_{a\,\mathrm{M}}$ (recall $\vec{T} = \prod_{i\in P} S_i \setminus Q_i$) we obtain $\pi_i(v) = \varepsilon$ for each $i \in a\,\mathrm{M}$. Hence, we have $\vec{q}'\restriction_{a\,\mathrm{M}} = \vec{p}\restriction_{a\,\mathrm{M}}$ and $va \approx_{\mathcal{A}} av$. This also implies $vaw_2 \approx_{\mathcal{A}} avw_2$. Using $\vec{p} \xrightarrow{a}_{\mathfrak{A}^{(k)}} \vec{r}$ we infer the following:

$$\vec{q}' = (\vec{p}\restriction_{a\,\mathrm{M}}, \vec{q}'\restriction_{P\setminus a\,\mathrm{M}}) \xrightarrow{a}_{\mathfrak{A}^{(k)}} \underbrace{(\vec{r}\restriction_{a\,\mathrm{M}}, \vec{q}'\restriction_{P\setminus a\,\mathrm{M}})}_{=:\vec{t}} \xrightarrow{[v]}_{\mathfrak{A}^{(0)}} (\vec{r}\restriction_{a\,\mathrm{M}}, \vec{p}\restriction_{P\setminus a\,\mathrm{M}}) = \vec{r}$$

(note that $\vec{t}$ and this av-labeled run from $\vec{q}'$ to $\vec{r}$ exist since $\mathfrak{A}^{(k)}$ is asynchronous). Due to $\vec{p}\restriction_{a\,\mathrm{M}} = \vec{q}'\restriction_{a\,\mathrm{M}}$ and $\vec{t}\restriction_{a\,\mathrm{M}} = \vec{r}\restriction_{a\,\mathrm{M}}$ we have $(\vec{q}', a, \vec{t}) \in \delta^{(k)}$. Hence, there is $\vec{q}'' \in \vec{S}$ with

$$(\vec{q}'\restriction_{a\,\mathrm{M}}, u, \vec{q}''\restriction_{a\,\mathrm{M}}) \in \Delta_a\,, \tag{9.3}$$

$\vec{q}'' \xrightarrow{[u^{\mathrm{R}}]}_{\mathfrak{A}^{(k-1)}} \vec{t}$, and $\vec{q}''\restriction_{a\,\mathrm{M}} \in \vec{Q}\restriction_{a\,\mathrm{M}}$. By asynchronism of $\mathfrak{S}$ and by Observation 9.5.5 we obtain also $u \leq a$ and $\vec{q}''\restriction_{P\setminus a\,\mathrm{M}} = \vec{q}'\restriction_{P\setminus a\,\mathrm{M}}$. Hence, we know $\vec{q}'' \in \vec{Q}$. Since we have $\vec{q}'' \xrightarrow{[u^{\mathrm{R}}vw_2]}_{\mathfrak{A}^{(k-1)}} \vec{s}$ we can apply our induction hypothesis yielding a state $\vec{q}''' \in \vec{Q}$ and a word $x \in A^*$ with

$$(\vec{q}''', [u^{\mathrm{R}}vw_2]) \to_{\mathfrak{S}}^* (\vec{s}, [x]) \tag{9.4}$$

and $\vec{q}''' \xrightarrow{[x]}_{\mathfrak{A}^{(0)}} \vec{s}$. Then we finally infer

$$\begin{aligned}(\vec{q}, [w]) &\overset{(9.1)}{=} (\vec{q}, [w_1 a w_2]) \xrightarrow{(9.2)}_{\mathfrak{S}}^* (\vec{q}', [vaw_2]) = (\vec{q}', [avw_2]) \\ &\xrightarrow{(9.3)}_{\mathfrak{S}} (\vec{q}'', [u^{\mathrm{R}}vw_2]) \xrightarrow{(9.4)}_{\mathfrak{S}}^* (\vec{q}''', [x])\,.\end{aligned}$$

Due to $\vec{q}''' \xrightarrow{[x]}_{\mathfrak{A}^{(0)}} \vec{s}$ we are done. ◁

Now, let $(\vec{q}, [w]) \in C(\mathfrak{A}^{(\infty)})$. By $C(\mathfrak{A}^{(\infty)}) = \bigcup_{k\in\mathbb{N}} C(\mathfrak{A}^{(k)})$ there is $k \in \mathbb{N}$ with $(\vec{q}, [w]) \in C(\mathfrak{A}^{(k)})$. Then we have $\vec{q} \xrightarrow{[w]}_{\mathfrak{A}^{(k)}} \vec{f}$ for a final state $\vec{f} \in F$. By the claim from above there are $\vec{p} \in \vec{Q}$ and $v \in A^*$ with $(\vec{q}, [w]) \to_{\mathfrak{S}}^* (\vec{p}, [v])$ and $\vec{p} \xrightarrow{[v]}_{\mathfrak{A}^{(0)}} \vec{f}$. In other words, we have $(\vec{q}, [w]) \in \mathrm{pre}_{\mathfrak{S}}^*(\{(\vec{p}, [v])\})$ and $(\vec{p}, [v]) \in C(\mathfrak{A}^{(0)})$. This finally implies $(\vec{q}, [w]) \in \mathrm{pre}_{\mathfrak{S}}^*(C(\mathfrak{A}^{(0)}))$. ◀

All in all, we have seen that $C(\mathfrak{A}^{(\infty)}) = \mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{A}^{(0)}))$ holds, i.e., this set of configurations is recognizable. So, we only have to show that $\mathfrak{A}^{(\infty)}$ can be constructed from $\mathfrak{A}^{(0)}$. To this end, recall that $\delta^{(0)} \subseteq \delta^{(1)} \subseteq \delta^{(2)} \subseteq \cdots \subseteq \delta^{(\infty)} \subseteq \vec{S} \times A \times \vec{S}$ holds. Since $\vec{S} \times A \times \vec{S}$ is a finite set, the sequence $(\delta^{(k)})_{k\in\mathbb{N}}$ stabilizes after at most $\ell := |\vec{S}|^2 \cdot |A|$ steps. Hence, we have $\mathfrak{A}^{(\ell)} = \mathfrak{A}^{(\infty)}$ which can be constructed as described above. This construction takes time $O(|\mathfrak{S}|^2 \cdot |\mathfrak{A}^{(0)}|^2 \cdot |A|)$ (similar to the construction from [BEM97]) and results in an $\mathfrak{S}$-asynchronous automaton having the same set of states as $\mathfrak{A}^{(0)}$ (however, there are possibly more transitions).

Remark 9.5.13. The inductive construction of $\mathfrak{A}^{(\infty)}$ as described above is not possible if $\mathfrak{A}^{(0)}$ is not asynchronous. To this end, let $(\vec{q}, \overline{a}t, \vec{p}) \in \Delta$ be a transition of $\mathfrak{S}$ and $b \in A$ with $a \parallel b$. Now, assume that $(\vec{p}, [t^{\mathrm{R}}bw])$ is accepted by an $\mathfrak{S}$-NFA $\mathfrak{B}$. Then we have $(\vec{q}, [abw]) \in \mathrm{pre}^*_{\mathfrak{S}}(C(\mathfrak{B}))$. Suppose that the only $[t^{\mathrm{R}}bw]$ accepting run of $\mathfrak{B}$ is the following one:

$$\vec{p} \xrightarrow{[b]}_{\mathfrak{B}} \vec{s'} \xrightarrow{[t^{\mathrm{R}}]}_{\mathfrak{B}} \vec{s} \xrightarrow{[w]}_{\mathfrak{B}} F\,.$$

Then we have to add a new path from $\vec{q}$ to $\vec{s}$ labeled with ab. To this end, we have to introduce one new state. Hence, the number of states of $\mathfrak{B}$ may increase in each iteration and, therefore, the aforementioned termination criterion is invalid.

In contrast, runs starting with some independent letters are not a problem if $\mathfrak{B}$ is asynchronous: since b-edges only modify the processes in $b\,\mathrm{M}$ and the $[t^{\mathrm{R}}]$-labeled run only affects the processes in $t\,\mathrm{M} \subseteq a\,\mathrm{M}$, there would be another run $\vec{p} \xrightarrow{[t^{\mathrm{R}}]}_{\mathfrak{B}} \vec{s''} \xrightarrow{[b]}_{\mathfrak{B}} \vec{s} \xrightarrow{[w]}_{\mathfrak{B}} F$ starting with $[t^{\mathrm{R}}]$. ⌟

Finally, from Theorem 9.5.9 we obtain the following decidability result:

Corollary 9.5.14. *The following reachability problem is decidable in polynomial time:*

Input: *A distributed alphabet $\mathcal{A}$, an asynchronous pushdown system $\mathfrak{S}$ over $\mathcal{A}$, and two $\mathfrak{S}$-asynchronous automata $\mathfrak{A}$ and $\mathfrak{B}$.*
Question: *Does $C(\mathfrak{A}) \to^*_{\mathfrak{S}} C(\mathfrak{B})$ hold?* ◂

9.5.2 Forwards Reachability

Next, we consider the sets of forwards reachable configurations in an asynchronous pushdown system. In the non-distributed case we know that, whenever we start from a regular set of configurations, we always reach another regular set of configurations (cf. Theorem 9.2.1). Unfortunately, this does not hold for systems having at least two (independent) processes. Actually, any rational trace language is forwards reachable in an asynchronous pushdown system starting from a very simple set of configurations. We prove this result in the following proposition:

Proposition 9.5.15. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $L \subseteq \mathbb{M}(\mathcal{A})$ be rational. Then there are a distributed alphabet $\mathcal{B} = (B, P, \mathrm{N})$ with $A \subseteq B$ and $\mathrm{N} \cap A \times P = \mathrm{M}$, an asynchronous pushdown system $\mathfrak{S} = (\vec{\boldsymbol{Q}}, \mathcal{P}_{\mathcal{B}}, \Delta, \Lambda)$ over $\mathcal{B}$, $c \in \mathrm{Conf}_{\mathfrak{S}}$, and $\vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}$ such that $\vec{\boldsymbol{q}}\,\mathrm{post}^*_{\mathfrak{S}}(\{c\}) \cap \mathbb{M}(\mathcal{A}) = L$.*

Proof. Since L is rational there is an NFA $\mathfrak{A} = (S, A, I, \delta, F)$ with $T(\mathfrak{A}) = L$. W.l.o.g., we can assume that $S \cap A = \varnothing$ holds. We want to simulate the runs of $\mathfrak{A}$ with the help of a (stateless) asynchronous pushdown system $\mathfrak{S}$ in inverse direction. To this end, our system initially writes a final state $f \in F$ on top of each stack of $\mathfrak{S}$. Then we simulate an edge $(p, a, q) \in \delta$ by reading the state q from all stacks and pushing an a to each stack $i \in a\,\mathrm{M}$ associated to a. Afterwards, we also push a p to all stacks. We are done, if $\mathfrak{A}$ is in an initial state. In this case, we simply remove such state $\iota \in I$ from all stacks.

For this construction we first have to extend our distributed alphabet $\mathcal{A}$ to $\mathcal{B} = (B, P, \mathrm{N})$ as follows:

- $B := A \cup S \cup \{\#\}$ where $\# \notin A \cup S$ and
- $\mathrm{N} := \mathrm{M} \cup ((Q \cup \{\#\}) \times P)$.

Then we construct an asynchronous pushdown system $\mathfrak{S} = (\vec{\boldsymbol{Q}}, \mathcal{P}_{\mathcal{B}}, \Delta, \Lambda)$ with:

- $\vec{\boldsymbol{Q}} := \{\vec{\mathsf{T}}\}$,
- $\Lambda\colon \vec{\boldsymbol{Q}} \to 2^{\mathrm{AP}}$ is arbitrary, and
- Δ consists of the following transitions:
  - $(\vec{\mathsf{T}}, \overline{\#}f, \vec{\mathsf{T}})$ for each $f \in F$,
  - $(\vec{\mathsf{T}}, \overline{q}ap, \vec{\mathsf{T}})$ for each transition $(p, a, q) \in \delta$, and
  - $(\vec{\mathsf{T}}, \overline{\iota}, \vec{\mathsf{T}})$ for each $\iota \in I$.

As mentioned before, $\mathfrak{S}$ simulates backwards computations of $\mathfrak{A}$. Underneath the top symbol $q \in S$ of our stacks, we find a trace λ such that $\mathfrak{A}$ has a w-labeled run (with $w \in \lambda$) from q to F. Formally, we have $(\vec{\mathsf{T}}, [p]) \to^*_{\mathfrak{S}} (\vec{\mathsf{T}}, [qw])$ if, and only if, we have $q \xrightarrow{w}_{\mathfrak{A}} p$ for each $p, q \in S$ and $w \in A^*$. This implies

$$(\vec{\mathsf{T}}, [\#]) \to^*_{\mathfrak{S}} (\vec{\mathsf{T}}, [w]) \iff [w] \in T(\mathfrak{A})$$

for each $w \in A^*$. Hence, we have $\vec{\mathsf{T}}\,\mathrm{post}^*_{\mathfrak{S}}(\{(\vec{\mathsf{T}}, [\#])\}) \cap \mathbb{M}(\mathcal{A}) = T(\mathfrak{A}) = L$. ◂

Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet with two distinct letters $a, b \in A$ satisfying $a \parallel b$, i.e., a and b are independent. Then the trace language $L := (ab)^*$ is rational, but not recognizable. Due to Proposition 9.5.15 we can find an asynchronous pushdown system $\mathfrak{S}$, a configuration $c \in \mathrm{Conf}_{\mathfrak{S}}$, and a state $\vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}$ such that the set of configurations with control state $\vec{\boldsymbol{q}}$ which are reachable from c have exactly the stack contents from L. Hence, the mapping $\mathrm{post}^*_{\mathfrak{S}}$ does not preserve recognizability. So, a natural question is to ask, to which class of configurations $\mathrm{post}^*_{\mathfrak{S}}(C)$ belongs for any recognizable set of configurations $C \subseteq \mathrm{Conf}_{\mathfrak{S}}$.

We can see that the membership problem of $\mathrm{post}^*_{\mathfrak{S}}(C)$ is decidable in polynomial time for any recognizable set C: let $c \in \mathrm{Conf}_{\mathfrak{S}}$. To check, whether $c \in \mathrm{post}^*_{\mathfrak{S}}(C)$ holds, we can compute an $\mathfrak{S}$-asynchronous automaton $\mathfrak{A}$ accepting $\mathrm{pre}^*_{\mathfrak{S}}(\{c\})$. Finally, we only have to check whether $C(\mathfrak{A}) \cap C$ is not empty. This is decidable in polynomial time due to the efficient closure properties of the class of recognizable trace languages. In other words, the membership problem of $\mathrm{post}^*_{\mathfrak{S}}(C)$ is in P.

We will see now that P is no optimal characterization of $\mathrm{post}^*_{\mathfrak{S}}(C)$. Concretely, we will see in the following theorem that the mapping $\mathrm{post}^*_{\mathfrak{S}} \colon 2^{\mathrm{Conf}_{\mathfrak{S}}} \to 2^{\mathrm{Conf}_{\mathfrak{S}}}$ effectively preserves rationality. Note that this is an even stronger result than Theorem 9.5.9 since we also characterize the reachable configurations from a proper superclass of the recognizable trace languages.

Theorem 9.5.16. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\mathfrak{S}$ be an asynchronous pushdown system, and $C \subseteq \mathrm{Conf}_{\mathfrak{S}}$ be rational. Then $\mathrm{post}^*_{\mathfrak{S}}(C)$ is rational. In particular, we can compute an $\mathfrak{S}$-NFA accepting $\mathrm{post}^*_{\mathfrak{S}}(C)$ from $\mathfrak{S}$ and an $\mathfrak{S}$-NFA accepting C. If $\mathcal{A}$ is fixed, this construction is possible in polynomial time.*

Let $\mathfrak{S} = (\vec{\boldsymbol{Q}}, \mathcal{P}_{\mathcal{A}}, \Delta, \Lambda)$ be an asynchronous pushdown system and $C \subseteq \mathrm{Conf}_{\mathfrak{S}}$ be rational. Our construction of an $\mathfrak{S}$-NFA accepting $\mathrm{post}^*_{\mathfrak{S}}(C)$ is inspired by the one by Finkel et al. [FWW97]. Concretely, the construction consists of the following three steps:

1. First, we construct from $\mathfrak{S}$ an equivalent *saturated* asynchronous pushdown system $\mathfrak{S}^{(\infty)}$. Such system satisfies the following condition: if $(\vec{\boldsymbol{p}}, \overline{a}t, \vec{\boldsymbol{q}}), (\vec{\boldsymbol{q}}, \overline{b}, \vec{\boldsymbol{r}}) \in \Delta$ with $t \approx_{\mathcal{A}} t'b$, then we also have $(\vec{\boldsymbol{q}}, \overline{a}s, \vec{\boldsymbol{r}}) \in \Delta$ for some $s \approx_{\mathcal{A}} t'$. This means, whenever we have a transition writing a letter b on top and another transition reading this letter afterwards, we can also apply both transitions at once.

   Now, consider a run of $\mathfrak{S}$ from $\vec{\boldsymbol{p}}$ to $\vec{\boldsymbol{q}}$ removing the letter a from the stacks' tops in a finite number of steps (i.e., $(\vec{\boldsymbol{p}}, [a]) \to^*_{\mathfrak{S}} (\vec{\boldsymbol{q}}, [\varepsilon])$). Then with the help of the described "shortcut" transitions we infer the existence of a transition $(\vec{\boldsymbol{p}}, \overline{a}, \vec{\boldsymbol{q}}) \in \Delta$.

   From such saturated asynchronous pushdown system we also obtain an asynchronous automaton $\mathfrak{E}$ simulating the read-only transitions[xvii]. Hence, we obtain $\vec{\boldsymbol{p}} \xrightarrow{[w]}_{\mathfrak{E}} \vec{\boldsymbol{q}}$ if, and only if, $(\vec{\boldsymbol{q}}, [\varepsilon]) \in \mathrm{post}^*_{\mathfrak{S}}(\{(\vec{\boldsymbol{p}}, [w])\})$.
2. Next, we consider an arbitrary run of our saturated system $\mathfrak{S}$. Then this run alternates between phases in which we decrease the distributed stack's height and phases in which we increase this height. Concretely, an increasing phase consists of transitions applying at least one write action while the decreasing phases consist of read-only transitions. We will prove that the effect of each such phase preserves rationality of the considered set of configurations efficiently.
3. Finally, we prove that the number of such phases of a run can be uniformly bounded. Note that in contrast to the non-distributed case this number is not necessarily 2. However, it only depends on the size of $\mathcal{A}$. Hence, we obtain an $\mathfrak{S}$-NFA accepting $\mathrm{post}^*_{\mathfrak{S}}(C)$ in polynomial time (if we consider $\mathcal{A}$ to be fixed).

[xvii] The letter $\mathfrak{E}$ indicates that this automaton accepts those traces which we are able to Erase from the top of $\mathfrak{S}$'s stacks.

Step 1. Saturate the System $\mathfrak{S}$

First, we want to saturate the asynchronous pushdown system $\mathfrak{S}$ by addition of several special "shortcut" transitions. For a non-distributed pushdown system, the idea was very simple: if there are two transitions $(p, \overline{a}tb, q)$ and $(q, \overline{b}, r)$ we simply add a new transition $(p, \overline{a}t, r)$. The addition of this new transition does not change the behavior of the pushdown system and transforms the system closer to a saturated one. In asynchronous pushdown systems, the technicalities are a bit more involved: suppose we have the two transitions $(\vec{p}, \overline{a}tbc, \vec{q})$ and $(\vec{q}, \overline{b}, \vec{r})$ with $b \parallel c$. Then we have $tbc \approx_{\mathcal{A}} tcb$, i.e., after doing the first transition the stacks from $b\,\mathrm{M}$ have the letter b on top and, hence, we can apply the second transition (eliminating b) immediately. Therefore, also in this situation, the addition of the transition $(\vec{p}, \overline{a}tc, \vec{r})$ helps to get closer to a saturated one.

Unfortunately, iterated elimination of letters from a transition may lead to an exponential blowup of our system. For example, let $a, b \in A$ be two independent letters (i.e., we have $a \parallel b$) and $\lambda_n := [(ab)^n]$ for $n \in \mathbb{N}$. The set of word suffixes of λ_n is $\{w \in \{a,b\}^* : |w|_a, |w|_b \leq n\}$. This language has size exponential in n. In contrast, the set of trace suffixes is $\{[a^k b^\ell] \mid k, \ell \leq n\}$ which has only quadratic size. In the general case, a trace $\lambda \in \mathbb{M}(\mathcal{A})$ has at most $O(|\lambda|^\alpha)$ many trace suffixes where α is the size of the maximal independent set in the dependence graph $\mathcal{G}_{\mathcal{A}}$ [BMS82].

Hence, for a transition $(\vec{p}, \overline{a}t, \vec{q})$ we should ensure to include for each trace suffix of $[t]$ at most one new transition. This can be done with the lexicographic normal form of the trace $[t]$:

Definition 9.5.17. Let $\mathcal{D} = (A, P, \mathrm{M})$ be a distributed alphabet, $\leq_{\mathrm{lex}}$ be a lexicographic ordering on A^*, and $u \in A^*$ be a word. Then the *lexicographic normal form* $\mathrm{lnf}(u)$ is the minimal word $v \in [u]$ wrt. $\leq_{\mathrm{lex}}$. ⌟

So, in the following we always ensure that for each transition $(\vec{p}, \overline{a}t, \vec{q})$ we have $t = \mathrm{lnf}(t)$. Concretely, we will now construct asynchronous pushdown systems $\mathfrak{S}^{(k)} := (\vec{\boldsymbol{Q}}, \mathcal{P}_{\mathcal{A}}, \Delta^{(k)}, \Lambda)$ for any $k \in \mathbb{N}$ as follows:

- we set $\Delta^{(0)} := \{(\vec{p}, \overline{a}\,\mathrm{lnf}(u), \vec{q}) \mid (\vec{p}, \overline{a}u, \vec{q}) \in \Delta\}$.
- to obtain $\Delta^{(k+1)}$, we add to the set $\Delta^{(k)}$ all transitions $(\vec{p}, \overline{a}\,\mathrm{lnf}(uv), \vec{r})$ for which there are a letter $b \in A$ and a state $\vec{r} \in \vec{\boldsymbol{Q}}$ with $(\vec{p}, \overline{a}ubv, \vec{r}), (\vec{r}, \overline{b}, \vec{q}) \in \Delta^{(k)}$ and $b \parallel v$ (i.e., we have $ubv \approx_{\mathcal{A}} uvb$ and, therefore, after application of the former transition there is a b on top of all stacks from $b\,\mathrm{M}$).

First, we want to check whether $\mathfrak{S}^{(k)}$ is really asynchronous. This can be done by induction on $k \in \mathbb{N}$. Let $k = 0$. For a transition $(\vec{p}, \overline{a}v, \vec{q}) \in \Delta^{(0)}$ there is another transition $(\vec{p}, \overline{a}u, \vec{q}) \in \Delta$ with $v = \mathrm{lnf}(u)$, i.e., $u \approx_{\mathcal{A}} v$. Since $\mathfrak{S}$ is asynchronous, we have $a\,\mathrm{M} \supseteq u\,\mathrm{M} = v\,\mathrm{M}$, i.e., $v \leq a$. Therefore, $\mathfrak{S}^{(0)}$ also is asynchronous.

Now, assume that $\mathfrak{S}^{(k)}$ is asynchronous. Let $(\vec{p}, \overline{a}ubv, \vec{r}), (\vec{r}, \overline{b}, \vec{q}) \in \Delta^{(k)}$ be two transitions with $b \parallel v$. Then from asynchronism of $\mathfrak{S}^{(k)}$ we know $ubv \leq a$ (recall that $x \leq y$ holds iff $x\,\mathrm{M} \subseteq y\,\mathrm{M}$). Additionally, we know $uv \leq ubv$ implying $uv \leq a$ by transitivity of the quasi-ordering $\leq$. Hence, $\overline{a}uv$ is a legal labeling of a transition. Since $uv \approx_{\mathcal{A}} \mathrm{lnf}(uv)$ and, hence, $[\![uv]\!] = [\![\mathrm{lnf}(uv)]\!]$ holds, also $\overline{a}uv$ is a legal labeling of a transition. Additionally, since the former transitions are independent of the processes from $P \setminus a\,\mathrm{M}$, the constructed ones are also independent of $P \setminus a\,\mathrm{M}$. Hence, $\mathfrak{S}^{(k+1)}$ also is asynchronous.

Let $\mathfrak{S}^{(\infty)} = (\vec{\boldsymbol{Q}}, \mathcal{P}_{\mathcal{A}}, Q^{(\infty)}, \Lambda)$ be the "limit" of our construction with $\Delta^{(\infty)} = \bigcup_{k \in \mathbb{N}} \Delta^{(k)}$. Then $\mathfrak{S}^{(\infty)}$ is saturated and still asynchronous. By construction we have $\Delta^{(0)} \subseteq \Delta^{(1)} \subseteq \Delta^{(2)} \subseteq \cdots \subseteq \Delta^{(\infty)}$. Since each $\Delta^{(k)}$ is finite (by finiteness of $|\Delta| = |\Delta^{(0)}|$) and since we only add transitions having a smaller number of write actions, this monotonically increasing sequence stabilizes. Hence, there is a number $\hat{k} \in \mathbb{N}$ with $\mathfrak{S}^{(\hat{k})} = \mathfrak{S}^{(\infty)}$. This number is $O(|u|^{\alpha} \cdot |\Delta|)$ where $u \in A^*$ is the longest word with $(\vec{\boldsymbol{p}}, \overline{a}u, \vec{\boldsymbol{q}}) \in \Delta$ (for $\vec{\boldsymbol{p}}, \vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}$ and $a \in A$) and α is the maximal size of an independent set in the dependence graph $\mathcal{G}_{\mathcal{A}}$ (by [BMS82]). Hence, the number $\hat{k}$ is polynomial in the size of $\mathfrak{S}$, but exponential in the size of $\mathcal{D}$. Therefore, if $\mathcal{D}$ is fixed, we can compute $\mathfrak{A}^{(\hat{k})} = \mathfrak{A}^{(\infty)}$ in polynomial time.

Example 9.5.18. Recall the asynchronous pushdown system $\mathfrak{S}$ from Example 9.5.4. In Figure 9.12 we depict our construction of $\mathfrak{S}^{(\infty)}$. ⌟

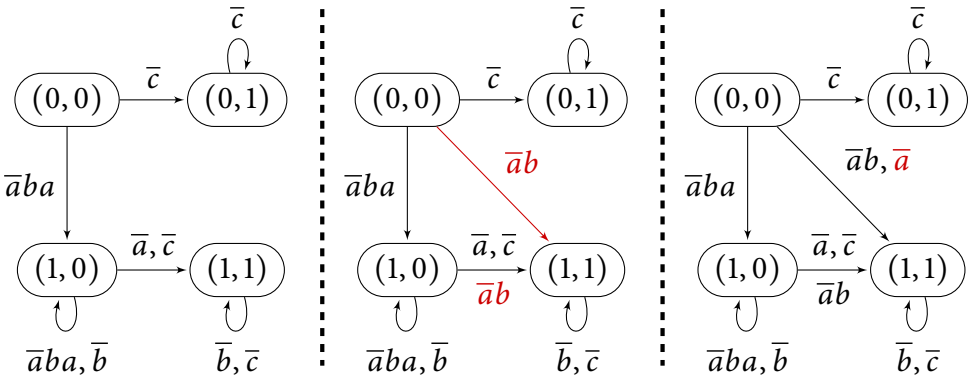

Figure 9.12. The asynchronous pushdown automata $\mathfrak{S} = \mathfrak{S}^{(0)}$, $\mathfrak{S}^{(1)}$, and $\mathfrak{S}^{(2)}$ (from left to right). The construction terminates after the second step. Hence, we obtain $\mathfrak{S}^{(2)} = \mathfrak{S}^{(\infty)}$ in this case.

Now, we want to show that the original asynchronous pushdown system $\mathfrak{S}$ and the constructed saturated system agree in their reachability relation. In other words, we prove that $\mathrm{post}^*_{\mathfrak{S}}(D) = \mathrm{post}^*_{\mathfrak{S}^{(\infty)}}(D)$ holds for any set $D \subseteq \mathrm{Conf}_{\mathfrak{S}} = \mathrm{Conf}_{\mathfrak{S}^{(\infty)}}$ of configurations. In particular, we will see that we have a read-only transition $(\vec{\boldsymbol{p}}, \overline{a}, \vec{\boldsymbol{q}})$ in $\Delta^{(\infty)}$ whenever we are able to remove an a from the top of the distributed stack on a run from $\vec{\boldsymbol{p}}$ to $\vec{\boldsymbol{q}}$.

Lemma 9.5.19. *Let $\vec{\boldsymbol{p}}, \vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}$, $a \in A$, $t \in A^*_{\leq a}$, and $k \in \mathbb{N}$. Then the following two statements hold:*

(1) If $(\vec{\boldsymbol{p}}, \overline{a}t, \vec{\boldsymbol{q}}) \in \Delta^{(k)}$ holds, we also have $(\vec{\boldsymbol{p}}, [a]) \to^*_{\mathfrak{S}} (\vec{\boldsymbol{q}}, [t^{\mathrm{R}}])$.

(2) If $(\vec{\boldsymbol{p}}, [a]) \to^*_{\mathfrak{S}} (\vec{\boldsymbol{q}}, [\varepsilon])$ holds, we also have $(\vec{\boldsymbol{p}}, \overline{a}, \vec{\boldsymbol{q}}) \in \Delta^{(\infty)}$.

Proof.

(1) We prove this statement by induction on $k \in \mathbb{N}$. First, let $k = 0$. Assume $(\vec{\boldsymbol{p}}, \overline{a}t, \vec{\boldsymbol{q}}) \in \Delta^{(0)}$. Then by definition of $\Delta^{(0)}$ there is $s \in A^*$ with $t = \mathrm{Inf}(s)$ (i.e., $[\![s]\!] = [\![t]\!]$) and $(\vec{\boldsymbol{p}}, \overline{a}s, \vec{\boldsymbol{q}}) \in \Delta$. Therefore, by definition of $\mathcal{M}(\mathfrak{S})$, we infer $(\vec{\boldsymbol{p}}, [a]) \to_{\mathfrak{S}} (\vec{\boldsymbol{q}}, [s^{\mathrm{R}}]) = (\vec{\boldsymbol{q}}, [t^{\mathrm{R}}])$.

Now, let $k > 0$. We know $\Delta^{(k-1)} \subseteq \Delta^{(k)}$. So, if $(\vec{p}, \overline{a}t, \vec{q}) \in \Delta^{(k-1)}$ holds, we are done by the induction hypothesis. We assume now that $(\vec{p}, \overline{a}t, \vec{q}) \in \Delta^{(k)} \setminus \Delta^{(k-1)}$ holds. Then there are $(\vec{p}, \overline{a}ubv, \vec{r}), (\vec{r}, \overline{b}, \vec{q}) \in \Delta^{(k-1)}$ with $b \parallel v$ and $t = \mathrm{lnf}(uv)$ (i.e., $[\![t]\!] = [\![uv]\!]$). By induction hypothesis we know $(\vec{p}, [a]) \to_{\mathfrak{S}}^* (\vec{r}, [(ubv)^{\mathrm{R}}])$ and $(\vec{r}, [b]) \to_{\mathfrak{S}}^* (\vec{q}, [\varepsilon])$. Then we infer

$$(p^{\mathrm{R}}, [a]) \to_{\mathfrak{S}}^* (\vec{r}, [(ubv)^{\mathrm{R}}]) = (\vec{r}, [b(uv)^{\mathrm{R}}]) \to_{\mathfrak{S}}^* (\vec{q}, [(uv)^{\mathrm{R}}]) = (\vec{q}, [t^{\mathrm{R}}]) .$$

(2) For each transition $(\vec{p}, \overline{a}t, \vec{q}) \in \Delta$ there is another transition $(\vec{p}, \overline{a}s, \vec{q}) \in \Delta^{(0)}$ of $\mathfrak{S}^{(0)}$ with $s = \mathrm{lnf}(t)$ and, hence, $[\![s]\!] = [\![t]\!]$. Therefore, from $(\vec{p}, [a]) \to_{\mathfrak{S}}^* (\vec{q}, [\varepsilon])$ we learn $(\vec{p}, [a]) \to_{\mathfrak{S}^{(0)}}^* (\vec{q}, [\varepsilon])$. Since additionally $\Delta^{(0)} \subseteq \Delta^{(\infty)}$ holds, we also have $(\vec{p}, [a]) \to_{\mathfrak{S}^{(\infty)}}^* (\vec{q}, [\varepsilon])$. Hence, there is a run in $\mathfrak{S}^{(\infty)}$ from $(\vec{p}, [a])$ to $(\vec{q}, [\varepsilon])$. We consider a shortest run Π in $\mathfrak{S}^{(\infty)}$ from $(\vec{p}, [a])$ to $(\vec{q}, [\varepsilon])$. We show next that this shortest run has length 1 implying $(\vec{p}, \overline{a}, \vec{q}) \in \Delta^{(\infty)}$.

Towards a contradiction, suppose the run Π has length at least 2. Then we apply at least one transition of the form $(\vec{r_1}, \overline{b}s, \vec{r_2})$ with $s \neq \varepsilon$ (i.e., we write at least one letter). We consider now the application of the last such transition writing at least one letter in Π. So, Π is the following run:

$$(\vec{p}, [a]) \to_{\mathfrak{S}^{(\infty)}}^* (\vec{r_1}, [w_1]) \xrightarrow{\overline{b}s}_{\mathfrak{S}^{(\infty)}} (\vec{r_2}, [w_2]) \xrightarrow{\overline{t}}_{\mathfrak{S}^{(\infty)}} (\vec{q}, [\varepsilon])$$

where $b \in A$, $s, t \in A^+$, and $(\vec{r_1}, \overline{b}s, \vec{r_2}) \in \Delta^{(\infty)}$. Since $s \neq \varepsilon$ there is $c \in A$ and $s' \in A^*$ with $s = s'c$. Then there is $v \in A^*$ with $[w_1] = [bv]$ and $[w_2] = [s^{\mathrm{R}}v] = [c\, s'^{\mathrm{R}}\, v]$. Additionally, by $(\vec{r_2}, [w_2]) \xrightarrow{\overline{t}}_{\mathfrak{S}^{(\infty)}} (\vec{q}, [\varepsilon])$ we know $t \in [w_2] = [cs'^{\mathrm{R}}v]$. Hence, there are $t_1, t_2 \in A^*$ with $t = t_1 c t_2$ (in the free monoid A^*) and $|t_1|_c = 0$ such that Π contains the following subrun

$$(\vec{r_2}, [w_2]) \xrightarrow{\overline{t_1}}_{\mathfrak{S}^{(\infty)}} (\vec{r_3}, [w_3]) \xrightarrow{\overline{c}}_{\mathfrak{S}^{(\infty)}} (\vec{r_4}, [w_4]) \xrightarrow{\overline{t_2}}_{\mathfrak{S}^{(\infty)}} (\vec{q}, [\varepsilon]) .$$

Since $t_1 c t_2 = t \approx_{\mathcal{A}} w_2 \approx_{\mathcal{A}} c\, s'^{\mathrm{R}}\, v$ and $|t_1|_c = 0$ we have $t_1 \parallel c$ in this case. Then, due to asynchronism of $\mathfrak{S}^{(\infty)}$ there is also a run labeled as follows

$$(\vec{r_2}, [w_2]) \xrightarrow{\overline{c}}_{\mathfrak{S}^{(\infty)}} (\vec{r_3'}, [w_3']) \xrightarrow{\overline{t_1}}_{\mathfrak{S}^{(\infty)}} (\vec{r_4}, [w_4]) \xrightarrow{\overline{t_2}}_{\mathfrak{S}^{(\infty)}} (\vec{q}, [\varepsilon])$$

and this modified run has the same length as Π. Hence, we have $(\vec{r_2}, \overline{c}, \vec{r_3'}) \in \Delta^{(\infty)}$. By $(\vec{r_1}, \overline{b}s, \vec{r_2}) \in \Delta^{(\infty)}$, $s = s'c$, and by construction of $\mathfrak{S}^{(\infty)}$ we also know $(\vec{r_1}, \overline{b}\,\mathrm{lnf}(s'), \vec{r_3'}) \in \Delta^{(\infty)}$. In other words, we can shorten Π by one edge as follows:

$$(\vec{p}, [a]) \to_{\mathfrak{S}^{(\infty)}}^* (\vec{r_1}, [w_1]) \to_{\mathfrak{S}^{(\infty)}} (\vec{r_3'}, [w_3']) \to_{\mathfrak{S}^{(\infty)}}^* (\vec{q}, [\varepsilon]) .$$

However, this is impossible since Π was a shortest run in $\mathfrak{S}^{(\infty)}$. Hence, Π has length 1 implying $(\vec{p}, \overline{a}, \vec{q}) \in \Delta^{(\infty)}$. ◂

Assumption 9.5.20. From now on, we assume that $\mathfrak{S}$ is saturated, i.e., that $\mathfrak{S} = \mathfrak{S}^{(\infty)}$ holds (and, hence, $\Delta = \Delta^{(\infty)}$). This is possible by $\mathrm{post}_{\mathfrak{S}}^*(D) = \mathrm{post}_{\mathfrak{S}^{(\infty)}}^*(D)$ for each $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$ (by Lemma 9.5.19) and since our construction is efficient (for a fixed distributed alphabet $\mathcal{A}$). ⌟

From this assumption and from Lemma 9.5.19 we also learn that $\mathfrak{S}$ is able to remove a letter a from the top of its stack on a run from $\vec{p}$ to $\vec{q}$ if, and only if, $\mathfrak{S}$ can do this in exactly one step. From this fact we obtain an asynchronous automaton $\mathfrak{E}$ which simulates exactly the read-only transitions of $\mathfrak{S}$. Concretely, this asynchronous automaton is $\mathfrak{E} = (\vec{Q}, \mathcal{A}, \vec{Q}, \delta_\varepsilon, \vec{Q})$ with

$$\delta_\varepsilon := \{(\vec{p}, a, \vec{q}) \mid (\vec{p}, \overline{a}, \vec{q}) \in \Delta\}.$$

Hence, we learn that the sequences of read-only transitions form a recognizable trace language.

The following statement sums up the semantics of the asynchronous automaton $\mathfrak{E}$:

Corollary 9.5.21. *Let $\vec{p}, \vec{q} \in \vec{Q}$ and $w \in A^*$. Then we have $\vec{p} \xrightarrow{[w]}_{\mathfrak{E}} \vec{q}$ if, and only if, we have $(\vec{p}, [w]) \to^*_{\mathfrak{S}} (\vec{q}, [\varepsilon])$.* ◂

Example 9.5.22. Recall Example 9.5.4. Then the asynchronous automaton $\mathfrak{E}$ of all read-only transitions in $\mathfrak{S}$ is depicted in Figure 9.13. For example, we see that $((0,0), [w]) \to^*_{\mathfrak{S}} ((1,1), [\varepsilon])$ holds if, and only if, $w \in a\{b,c\}^*$ holds. ⌟

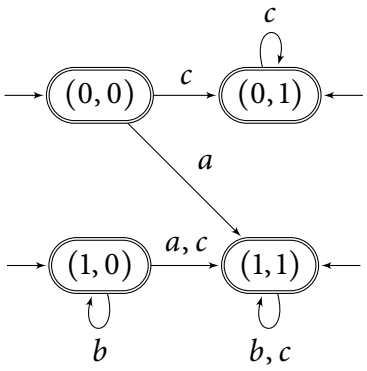

Figure 9.13. The asynchronous automaton $\mathfrak{E}$ derived from $\mathfrak{S}$.

Step 2. Analyzing the Phases of Runs in $\mathfrak{S}$

As previously announced we want to find an alternation of two types of phases in each run in the saturated asynchronous pushdown system $\mathfrak{S}$. Concretely, either $\mathfrak{S}$ decreases the height of its distributed stack or it increases this height. We consider these two kinds of phases separately.

To this end, we will partition the transition relation Δ of $\mathfrak{S}$ into multiple sets of transitions resulting in some "homogeneous" asynchronous pushdown subsystems of $\mathfrak{S}$:

- $\Delta_\varepsilon := \Delta \cap (\vec{Q} \times \overline{A} \times \vec{Q})$. Set $\mathfrak{S}_\varepsilon = (\vec{Q}, \mathcal{P}_{\mathcal{A}}, \Delta_\varepsilon, \Lambda)$.
- $\Delta_{=a} := \Delta \cap (\vec{Q} \times \overline{A_{=a}} A^+ \times \vec{Q})$ for $a \in A$. Set $\mathfrak{S}_{=a} = (\vec{Q}, \mathcal{P}_{\mathcal{A}}, \Delta_{=a}, \Lambda)$.

First we consider the effect of $\mathfrak{S}_\varepsilon$ which simulates the decreasing phases of the runs in $\mathfrak{S}$. So, for a set $D \subseteq \mathrm{Conf}_{\mathfrak{S}} = \mathrm{Conf}_{\mathfrak{S}_\varepsilon}$ of configurations we first define the following set of configurations with decreased stack heights:

$$\mathrm{R}(D) := \{c \in \mathrm{Conf}_{\mathfrak{S}} \mid D \to^*_{\mathfrak{S}_\varepsilon} c\} = \mathrm{post}^*_{\mathfrak{S}_\varepsilon}(D)\,.$$

We should note here, that the asynchronous pushdown system $\mathfrak{S}_\varepsilon$ corresponds to our asynchronous automaton $\mathfrak{E}$ which we have constructed in the previous step. Concretely, we have $\vec{p} \xrightarrow{[w]}_{\mathfrak{E}} \vec{q}$ if, and only if, $(\vec{p}, [w]) \to^*_{\mathfrak{S}_\varepsilon} (\vec{q}, [\varepsilon])$ holds. Hence, when computing the reachable configurations via $\mathfrak{S}_\varepsilon$, we will use $\mathfrak{E}$ instead. Since $\mathfrak{E}$ is asynchronous, its accepted trace language is recognizable in $\mathbb{M}(\mathcal{A})$. Therefore, we obtain the rationality of $\mathrm{R}(D)$ from a rational set $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$ of configurations using the efficient closure properties of rational and recognizable trace languages (cf. Appendix A).

Lemma 9.5.23. *Let $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$ be a rational set of configurations. Then the set $\mathrm{R}(D)$ is rational. In particular, we can compute an $\mathfrak{S}$-NFA accepting $\mathrm{R}(D)$ from $\mathfrak{S}$ and an $\mathfrak{S}$-NFA accepting D. If $\mathcal{A}$ is fixed, this construction is possible in polynomial time.*

Proof. Let $\mathfrak{A} = (S, A, \vec{\boldsymbol{Q}}, \delta, F)$ be an $\mathfrak{S}$-NFA with $C(\mathfrak{A}) = D$. We show that $\mathrm{R}(D)$ is essentially a left-quotient of D wrt. recognizable trace languages of read action sequences. Concretely, we show that

$$\mathrm{R}(D) = \bigcup_{\vec{p},\vec{q} \in \vec{\boldsymbol{Q}}} \{\vec{q}\} \times_{T(\mathfrak{E}_{\vec{p}\to\vec{q}})} \backslash T(\mathfrak{A}_{\vec{p}\to F})$$

holds. Since the class of rational trace languages is efficiently closed under union and left-quotients wrt. recognizable trace languages (cf. Theorem A.2, recall that $\mathfrak{E}$ is an asynchronous automaton), the right-hand side of this equation is efficiently rational (for a fixed $\mathcal{A}$). For better readability we denote this big union by X.

First, we prove the inclusion "$\subseteq$". To this end, let $(\vec{q}, \lambda) \in \mathrm{R}(D)$. Then by definition there are $(\vec{p}, \kappa) \in D$ (i.e., $\kappa \in T(\mathfrak{A}_{\vec{p}\to F})$) and $r \in A^*$ with $(\vec{p}, \kappa) \xrightarrow{\overline{r}}_{\mathfrak{S}_\varepsilon} (\vec{q}, \lambda)$. We know, by the semantics of $\overline{r}$, that $\kappa = [r] \cdot \lambda$ holds. In particular, we have $(\vec{p}, [r]) \xrightarrow{\overline{r}}_{\mathfrak{S}_\varepsilon} (\vec{q}, [\varepsilon])$. By Corollary 9.5.21 we obtain $\vec{p} \xrightarrow{[r]}_{\mathfrak{E}} \vec{q}$, i.e., $[r] \in T(\mathfrak{E}_{\vec{p}\to\vec{q}})$. This implies

$$(\vec{q}, \lambda) = (\vec{q}, {}_{[r]}\backslash\kappa) \in \{\vec{q}\} \times_{T(\mathfrak{E}_{\vec{p}\to\vec{q}})} \backslash T(\mathfrak{A}_{\vec{p}\to F}) \subseteq X\,.$$

Towards the converse inclusion, let $(\vec{q}, \lambda) \in X$. Then there exists a control state $\vec{p} \in \vec{\boldsymbol{Q}}$ with $\lambda \in {}_{T(\mathfrak{E}_{\vec{p}\to\vec{q}})}\backslash T(\mathfrak{A}_{\vec{p}\to F})$. There are $[r] \in T(\mathfrak{E}_{\vec{p}\to\vec{q}})$ and $\kappa \in T(\mathfrak{A}_{\vec{p}\to F})$ (i.e., $(\vec{p}, \kappa) \in D$) with $\lambda = {}_{[r]}\backslash\kappa$, i.e., $\kappa = [r] \cdot \lambda$. From Corollary 9.5.21 we infer $(\vec{p}, [r]) \xrightarrow{\overline{r}}_{\mathfrak{S}_\varepsilon} (\vec{q}, [\varepsilon])$. Hence,

$$D \ni (\vec{p}, \kappa) = (\vec{p}, [r] \cdot \lambda) \xrightarrow{\overline{r}}_{\mathfrak{S}_\varepsilon} (\vec{q}, [\varepsilon] \cdot \lambda) = (\vec{q}, \lambda)$$

holds implying $(\vec{q}, \lambda) \in \mathrm{R}(D)$. ◂

Next, we consider the effect of the asynchronous pushdown systems $\mathfrak{S}_{=a}$ where $a \in A$. These systems simulate the increasing phases of the runs in $\mathfrak{S}$. For a set $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$ of configurations we define the set of configurations with increased stack heights as follows:

$$\mathrm{W}(D) := \{c \in \mathrm{Conf}_{\mathfrak{S}} \mid \exists a \in A\colon D \to^*_{\mathfrak{S}_{=a}} c\} = \bigcup_{a \in A} \mathrm{post}^*_{\mathfrak{S}_{=a}}(D)\,.$$

Now, we want to proof that the W-operation preserves rationality efficiently. Since the class of rational trace languages is closed under finite union, it suffices to show that $\mathrm{post}^*_{\mathfrak{S}_{=a}}(D)$ is rational for any $a \in A$. So, let $a \in A$. By definition all transitions of $\mathfrak{S}_{=a}$ read only letters from $A_{=a}$, i.e., letters associated to exactly the processes $a\,\mathrm{M}$. So, we can understand these read actions as some kind of synchronization of the involved processes. Therefore, we can also see $\mathfrak{S}_{=a}$ as a (non-distributed) single-pushdown system. Additionally, by definition no transition of $\mathfrak{S}_{=a}$ ever decreases the height of the distributed stack (since we require that each transition writes at least one letter). Hence, to compute $\mathrm{post}^*_{\mathfrak{S}_{=a}}(D)$ it suffices to consider only the case where the distributed stack contains exactly one letter $b \in A_{=a}$. By Theorem 9.2.1 we already know for such systems how to compute $\mathrm{post}^*_{\mathfrak{S}_{=a}}(\{(\vec{\boldsymbol{p}}, b)\})$ (where the stack contents are seen as words). Then we obtain $\mathrm{W}(D)$ by removing the top letter b (this corresponds to taking a left quotient) and prepending $\eta(\mathrm{post}^*_{\mathfrak{S}_{=a}}(\{(\vec{\boldsymbol{p}}, b)\}))$.

Lemma 9.5.24. *Let $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$ be a rational set of configurations. Then the set $\mathrm{W}(D)$ is rational. We can construct an $\mathfrak{S}$-NFA accepting $\mathrm{W}(D)$ from $\mathfrak{S}$ and an $\mathfrak{S}$-NFA accepting D. In particular, this construction is possible in polynomial time.*

Proof. Let $\mathfrak{A} = (S, A, \vec{\boldsymbol{Q}}, \delta, F)$ be an $\mathfrak{S}$-NFA with $C(\mathfrak{A}) = D$. Let $a \in A$ be an arbitrary letter. We can understand the asynchronous pushdown system $\mathfrak{S}_{=a} = (\vec{\boldsymbol{Q}}, \mathcal{P}_{\mathcal{A}}, \Delta_{=a}, \Lambda)$ as a (non-distributed) single-pushdown system $\mathfrak{T}_{=a} = (\vec{\boldsymbol{Q}}, \mathcal{P}_A, \Delta_{=a}, \Lambda)$ (note that $\mathfrak{S}_{=a}$ and $\mathfrak{T}_{=a}$ only differ in their underlying data type). Then from a state $\vec{\boldsymbol{p}} \in \vec{\boldsymbol{Q}}$, and $\mathfrak{T}_{=a}$ we can compute a $\mathfrak{T}_{=a}$-NFA $\mathfrak{B}^{\vec{\boldsymbol{p}},a} = (S^{\vec{\boldsymbol{p}},a}, A, \vec{\boldsymbol{Q}}, \delta^{\vec{\boldsymbol{p}},a}, F^{\vec{\boldsymbol{p}},a})$ accepting $\mathrm{post}^*_{\mathfrak{T}_{=a}}(\{(\vec{\boldsymbol{p}}, a)\}) \subseteq \vec{\boldsymbol{Q}} \times A^*$ according to Theorem 9.2.1. Note that for any $\vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}$ the language $L\big(\mathfrak{B}^{\vec{\boldsymbol{p}},a}_{\vec{\boldsymbol{q}} \to F^{\vec{\boldsymbol{p}},a}}\big)$ is regular and, hence, $T\big(\mathfrak{B}^{\vec{\boldsymbol{p}},a}_{\vec{\boldsymbol{q}} \to F^{\vec{\boldsymbol{p}},a}}\big) = \eta\big(L\big(\mathfrak{B}^{\vec{\boldsymbol{p}},a}_{\vec{\boldsymbol{q}} \to F^{\vec{\boldsymbol{p}},a}}\big)\big)$ is rational in $\mathbb{M}(\mathcal{A})$.

We prove next that the following equation holds:

$$\mathrm{W}(D) = \bigcup_{a \in A} \bigcup_{\vec{\boldsymbol{p}}, \vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}} \{\vec{\boldsymbol{q}}\} \times \Big(T\big(\mathfrak{B}^{\vec{\boldsymbol{p}},a}_{\vec{\boldsymbol{q}} \to F^{\vec{\boldsymbol{p}},a}}\big) \cdot {}_a\backslash T(\mathfrak{A}_{\vec{\boldsymbol{p}} \to F})\Big)\,.$$

Since the rational trace languages are efficiently closed under union, concatenation, and left-quotients wrt. recognizable trace languages, the right-hand side of this equation is efficiently rational. For better readability we denote this big union by X.

First, we assume that $(\vec{\boldsymbol{q}}, \lambda) \in \mathrm{W}(D)$ holds. Then there is $a \in A$ and $(\vec{\boldsymbol{p}}, \kappa) \in D$ with $(\vec{\boldsymbol{p}}, \kappa) \to^*_{\mathfrak{S}_{=a}} (\vec{\boldsymbol{q}}, \lambda)$ and $\kappa = [a] \cdot \kappa' \in T(\mathfrak{A}_{\vec{\boldsymbol{p}} \to F})$ (where $\kappa' \in \mathbb{M}(\mathcal{A})$). Since $\mathfrak{S}_{=a}$ only reads letters from $A_{=a}$ and never decreases the stack's height, it will never touch a letter below the a on top. Hence, κ' also is a suffix of λ. So, there is a trace $\mu \in \mathbb{M}(\mathcal{A})$ with $(\vec{\boldsymbol{p}}, [a]) \to^*_{\mathfrak{S}_{=a}} (\vec{\boldsymbol{q}}, \mu)$ and $\lambda = \mu\kappa'$. Then there also is a word $w \in \mu$ with $(\vec{\boldsymbol{p}}, a) \to^*_{\mathfrak{T}_{=a}} (\vec{\boldsymbol{q}}, w)$ implying $(\vec{\boldsymbol{q}}, w) \in$

$\mathrm{post}^*_{\mathfrak{T}_{=a}}(\{(\vec{p}, a)\}) = C(\mathfrak{B}^{\vec{p},a})$ and $w \in L(\mathfrak{B}^{\vec{p},a}_{\vec{q}\to F\vec{p},a})$. Then we infer $\mu = [w] \in T(\mathfrak{B}^{\vec{p},a}_{\vec{q}\to F\vec{p},a})$ and, finally,

$$\lambda = \mu\kappa' = \mu \cdot {}_a\backslash\kappa \in T(\mathfrak{B}^{\vec{p},a}_{\vec{q}\to F\vec{p},a}) \cdot {}_a\backslash T(\mathfrak{A}_{\vec{p}\to F}),$$

i.e., $(\vec{q}, \lambda) \in X$.

Towards the converse inclusion, let $(\vec{q}, \lambda) \in X$. Then there is $a \in A$ and $\vec{p} \in \vec{Q}$ with $\lambda \in T\big(\mathfrak{B}^{\vec{p},a}_{\vec{q}\to F\vec{p},a}\big) \cdot {}_a\backslash T(\mathfrak{A}_{\vec{p}\to F})$. So, we can factorize λ as follows: $\lambda = \lambda_1\lambda_2$ where $\lambda_1 \in T\big(\mathfrak{B}^{\vec{p},a}_{\vec{q}\to F\vec{p},a}\big)$ and $[a] \cdot \lambda_2 \in T(\mathfrak{A}_{\vec{p}\to F})$. From the latter statement we learn $(\vec{p}, [a]\lambda_2) \in D$. From the former one we obtain a word $w \in \lambda_1$ with $w \in L\big(\mathfrak{B}^{\vec{p},a}_{\vec{q}\to F\vec{p},a}\big)$ implying

$$(\vec{q}, w) \in C(\mathfrak{B}^{\vec{p},a}) = \mathrm{post}^*_{\mathfrak{T}_{=a}}(\{(\vec{p}, a)\}).$$

Hence, we also have $(\vec{q}, \lambda_1) = (\vec{q}, [w]) \in \mathrm{post}^*_{\mathfrak{S}_{=a}}(\{(\vec{p}, [a])\})$. But this also implies

$$D \ni (\vec{p}, [a] \cdot \lambda_2) \to^*_{\mathfrak{S}_{=a}} (\vec{q}, \lambda_1 \cdot \lambda_2) = (\vec{q}, \lambda)$$

and, therefore, $(\vec{q}, \lambda) \in \mathrm{post}^*_{\mathfrak{S}_{=a}}(D) \subseteq \mathrm{W}(D)$. ◂

It is a simple task to prove that any run of $\mathfrak{S}$ only consists of a finite number of phases in which we either decrease or increase the distributed stack's height. Moreover, since we allow these phases to be empty, we can assume that each run alternates between these two phases. Hence, we obtain $\mathrm{post}^*_{\mathfrak{S}}(C)$ by any number of alternations of the R- and W-operations:

Proposition 9.5.25. *Let $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$ be a set of configurations. Then the following statements hold:*

(1) For each $c \in \mathrm{post}^*_{\mathfrak{S}}(D)$ there exists $\ell \in \mathbb{N}$ with $c \in (\mathrm{RW})^\ell(D)$.

(2) For each $\ell \in \mathbb{N}$ we have $(\mathrm{RW})^\ell(D) \subseteq \mathrm{post}^*_{\mathfrak{S}}(D)$.

In particular, we have $\mathrm{post}^*_{\mathfrak{S}}(D) = (\mathrm{RW})^*(D)$. ◂

Step 3. Uniformly Bounding the Number of Phases

By Proposition 9.5.25 we know that each run of $\mathfrak{S}$ can be split into a finite number of phases alternating between decreasing and increasing the stack's height. If our system $\mathfrak{S}$ has only one pushdown, we know that each reachable configuration can be reached via a run having at most two of such phases [FWW97]. However, as the following example shows, for multi-pushdown systems we cannot bound the number of phases to two:

Example 9.5.26. Let $\mathcal{A} = (A, P, \mathrm{M})$ be the distributed alphabet with $A = \{a, b, c, d, e\}$ and $P = \{1, 2, 3\}$ where $a\,\mathrm{M} = P$, $b\,\mathrm{M} = \{1, 2\}$, $c\,\mathrm{M} = \{3\}$, $d\,\mathrm{M} = \{1\}$, and $e\,\mathrm{M} = \{2\}$. Further let $\mathfrak{S} = (\vec{Q}, \mathcal{P}_{\mathcal{A}}, \Delta, \Lambda)$ be the asynchronous pushdown system from Figure 9.14.

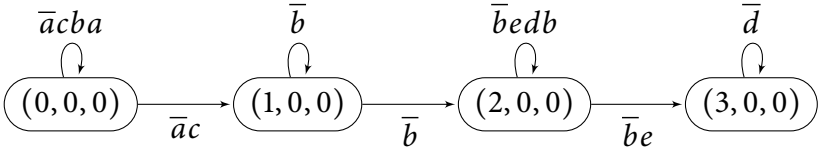

Figure 9.14. The asynchronous pushdown system from Example 9.5.26.

One can check that $\mathfrak{S}$ is saturated. The following is the only run from the configuration $((0,0,0),[a])$ to $((3,0,0),[c^4e^4])$:

$$\begin{aligned}((0,0,0),[a]) &\to_{\mathfrak{S}}^{3} ((0,0,0),[ab^3c^3]) \to_{\mathfrak{S}} ((1,0,0),[b^3c^4])\\ &\to_{\mathfrak{S}}^{2} ((2,0,0),[bc^4])\\ &\to_{\mathfrak{S}}^{3} ((2,0,0),[bc^4d^3e^3]) \to_{\mathfrak{S}} ((3,0,0),[c^4d^3e^4])\\ &\to_{\mathfrak{S}}^{3} ((3,0,0),[c^4e^4])\,.\end{aligned}$$

Note that this run splits into four phases (that correspond to the four lines above). It increases its pushdowns in the first and third and decreases them in the second and fourth line. ⌟

However, we will prove now that there is still a uniform bound of the number of alternations between decreasing and increasing phases for any run of a saturated asynchronous pushdown system $\mathfrak{S}$. This means, there is a number ℓ (which is linear in the size of $\mathcal{A}$) such that for any given forwards reachable configuration there is a run in $\mathfrak{S}$ from C alternating at most ℓ times between decreasing and increasing phases.

For technical reasons we have to introduce the following notations:

(1) Let $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$ be a set of configurations and $a \in A$ be a letter. We write $\mathrm{RW}_{=a}(D)$ for the set of configurations $\mathrm{post}^{+}_{\mathfrak{S}_{=a}}(\mathrm{post}^{*}_{\mathfrak{S}_{\varepsilon}}(D))$. Note that we require the phase increasing the distributed stack's height to be non-empty.

(2) Let $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$ be a set of configurations, $w \in A^*$ be a word, and $a \in A$ be a letter. Then we inductively define the following sets of configurations:

$$\mathrm{RW}_{=\varepsilon}(D) := D \qquad \text{and} \qquad \mathrm{RW}_{=aw} := \mathrm{RW}_{=w}(\mathrm{RW}_{=a}(D))\,.$$

It is easy to see that the following two equations hold:

$$\mathrm{RW}(D) = \mathrm{R}(D) \cup \bigcup_{a\in A} \mathrm{RW}_{=a}(D) \quad \text{and} \quad (\mathrm{RW})^*(D) = \bigcup_{w\in A^*} \mathrm{RW}_{=w}\,\mathrm{R}(D)$$

for any $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$. Until now $\mathrm{post}^{*}_{\mathfrak{S}}(D) = (\mathrm{RW})^*(D)$ is still an infinite union of rational sets of configurations. However, the next lemma states that the sets $\mathrm{RW}_{=w}(D)$ for long words $w \in A^*$ are also included in sets $\mathrm{RW}_{=v}(D)$ for shorter words $v \in A^*$.

Lemma 9.5.27. *Let $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$ be a set of configurations and $a_1, \ldots, a_n, a_{n+1} \in A$ be some letters with the following properties $a_1 \leq a_{n+1}$ and $a_k \not\leq a_{n+1}$ for each $1 < k \leq n$. Then we have:*

$$\mathrm{RW}_{=a_1a_2\ldots a_n a_{n+1}}(D) \subseteq \mathrm{RW}_{=a_2\ldots a_n a_{n+1}}(D)\,.$$

Proof. Let $c, d \in \mathrm{Conf}_{\mathfrak{S}}$ be two configurations with $d \in \mathrm{RW}_{=a_1 a_2 \dots a_n a_{n+1}}(c)$. Consider a run from c to d of $\mathfrak{S}$ which can be split into such phases as described by $\mathrm{RW}_{=a_1 a_2 \dots a_n a_{n+1}}$. We first show that the number of $\mathfrak{S}_{=a_1}$-transitions in this run can be reduced by one. Afterwards, induction on the number of $\mathfrak{S}_{=a_1}$-transition yields our claim.

▷ *Claim 1.* *Let $a \in A_{=a_1}$, $x \in A^*$, and $t \in \prod_{k=2}^{n+1} \overline{A}^* (\overline{A}_{=a_k} A^+)^+$ with $(\vec{p}, [u]) \xrightarrow{\overline{a}x}_{\mathfrak{S}} (\vec{q}, [v]) \xrightarrow{t}_{\mathfrak{S}} (\vec{r}, [w])$. Then there is another action sequence $s \in \prod_{k=2}^{n+1} \overline{A}^* (\overline{A}_{=a_k} A^+)^+$ with $(\vec{p}, [u]) \xrightarrow{s}_{\mathfrak{S}} (\vec{r}, [w])$.*

Proof. We show this statement by induction on the length $|x|$ of x (the word we push onto the distributed stack in the first transition). If $x = \varepsilon$ holds, we can set $s := \overline{a}t \in \prod_{k=2}^{n+1} \overline{A}^* (\overline{A}_{=a_k} A^+)^+$. Then we are done since $(\vec{p}, [u]) \xrightarrow{s}_{\mathfrak{S}} (\vec{r}, [w])$ holds by assumption. Now, let $x = x'b$ for a word $x' \in A^*$ and $b \in A$, i.e., we have an b on top of the stacks with indices in $b\,\mathrm{M}$ after application of the first transition. Note that $b \leq a$ and, therefore, $b \leq a \leq a_1 \leq a_{n+1}$ by asynchronism of $\mathfrak{S}$.

Let $\vec{q_k} \in \vec{Q}$, $c_k \in A$, $y_k, z_k \in A^*$ (for $0 \leq k \leq \ell$) such that

- $(\vec{q}, [v]) = (\vec{q_0}, [c_0 z_0])$,
- $(\vec{q_k}, \overline{c_k} y_k, \vec{q_{k+1}}) \in \Delta$ implying $(\vec{q_k}, [c_k z_k]) \to_{\mathfrak{S}} (\vec{q_{k+1}}, [{y_k}^{\mathrm{R}} z_k])$ for each $0 \leq k < \ell$,
- $[{y_{k-1}}^{\mathrm{R}} z_{k-1}] = [c_k z_k]$ for each $1 \leq k \leq \ell$,
- $(\vec{r}, [w]) = (\vec{q_\ell}, [{y_\ell}^{\mathrm{R}} z_\ell])$, and
- $t = \overline{c_0} y_0 \overline{c_1} y_1 \dots \overline{c_\ell} y_\ell$.

In particular, we have $c_\ell \in A_{=a_{n+1}}$ by the definition of t. Hence, we learn

$$b\,\mathrm{M} \subseteq x\,\mathrm{M} \subseteq a\,\mathrm{M} = a_1\,\mathrm{M} \subseteq a_{n+1}\,\mathrm{M} = c_\ell\,\mathrm{M}\ .$$

Consequently, there is $0 \leq k < \ell$ with $b \leq c_k$. We choose this number k minimal with $b \leq c_k$. Next, we show the following two properties:

(1) $b = c_k$.
From $b \leq c_k$ we obtain $b\,\mathrm{M} \cap c_k\,\mathrm{M} \neq \varnothing$ implying the existence of $0 \leq j \leq k$ with $b\,\mathrm{M} \cap c_j\,\mathrm{M} \neq \varnothing$. We choose j minimal with this property. Then $(\vec{q_j}, \overline{c_j} y_j, \vec{q_{j+1}}) \in \Delta$ is the first transition of this run touching any of the stacks with index in $b\,\mathrm{M}$. Hence, the stacks from $b\,\mathrm{M}$ in configuration $(\vec{q_j}, [c_j z_j])$ coincide with those in $(\vec{q}, [v]) = (\vec{q_0}, [c_0 z_0])$ and, therefore, still have the letter b on their top position. On the other hand, all stacks from $a_j\,\mathrm{M}$ have the letter a_j on top since the transition $(\vec{q_j}, \overline{c_j} y_j, \vec{q_{j+1}}) \in \Delta$ can be applied. Then from $b\,\mathrm{M} \cap c_j\,\mathrm{M} \neq \varnothing$ we learn $b = c_j$ implying in particular $b \leq c_j$. Since $0 \leq k < \ell$ was minimal with $b \leq c_k$ we get $j = k$ and, hence, $b = c_k$.

(2) $b \parallel c_j$ for each $0 \leq j < k$.
The choice of j above implies that, for all $0 \leq j < k$, we have $\varnothing = b\,\mathrm{M} \cap c_j\,\mathrm{M}$ and, therefore, $b \parallel c_j$.

Now, we have to distinguish two cases, namely whether the word y_k is empty or not. First, suppose $y_k = \varepsilon$. By asynchronism we know $y_j \leq c_j$ implying $b \parallel y_j$ for each $0 \leq j < k$. Therefore, we learn $\overline{c_k} = \overline{b} \parallel \overline{c_0} y_0 \overline{c_1} y_1 \dots \overline{c_{k-1}} y_{k-1}$. So, according to Observation 9.5.6 we are able to reorder the transitions of our run from $(\vec{p}, [u])$ to $(\vec{r}, [w])$ as follows:

- first apply the *global* transition $(\vec{p}, \overline{a}x, \vec{q})$,

- then do the *local* transition $(\vec{\boldsymbol{q}}_{\boldsymbol{k}}{\restriction_{c_k\,\mathrm{M}}}, \overline{c_k}, \vec{\boldsymbol{q}}_{\boldsymbol{k+1}}{\restriction_{c_k\,\mathrm{M}}})$,
- then follow the *local* transitions $(\vec{\boldsymbol{q}}_{\boldsymbol{j}}{\restriction_{c_j\,\mathrm{M}}}, \overline{c_j} y_j \vec{\boldsymbol{q}}_{\boldsymbol{j+1}}{\restriction_{c_j\,\mathrm{M}}})$ for each $0 \leq j < k$, and
- finally follow the original run from $(\vec{\boldsymbol{q}}_{\boldsymbol{k+1}}, [c_{k+1} z_{k+1}])$ to $(\vec{\boldsymbol{r}}, [w])$.

Hence, we have reordered the run from $(\vec{\boldsymbol{p}}, [u])$ to $(\vec{\boldsymbol{r}}, [w])$ such that the second local transition (starting in $(\vec{\boldsymbol{q}}, [v])$) is $(\vec{\boldsymbol{q}}_{\boldsymbol{k}}{\restriction_{b\,\mathrm{M}}}, \overline{b}, \vec{\boldsymbol{q}}_{\boldsymbol{k+1}}{\restriction_{b\,\mathrm{M}}})$. Note that this reordered run still has the same properties as the original path labeled with $\overline{a} x t$. So, without loss of generality, we can assume that $k = 0$ holds. Then we have the following two consecutive transitions:

$$(\vec{\boldsymbol{p}}, \overline{a} x, \vec{\boldsymbol{q}}) \quad \text{and} \quad (\vec{\boldsymbol{q}}, \overline{b}, \vec{\boldsymbol{q}}_{\boldsymbol{1}})$$

in the saturated system $\mathfrak{S}$. Then saturation implies the existence of the transition $(\vec{\boldsymbol{p}}, \overline{a} x', \vec{\boldsymbol{q}}_{\boldsymbol{1}}) \in \Delta$. Hence, we have

$$(\vec{\boldsymbol{p}}, [v]) \xrightarrow{\overline{a} x'}_{\mathfrak{S}} (\vec{\boldsymbol{q}}_{\boldsymbol{1}}, [c_1 z_1]) \xrightarrow{\overline{c_1} y_1 \ldots \overline{c_{k-1}} y_{k-1} \overline{c_{k+1}} y_{k+1} \ldots \overline{c_\ell} y_\ell}_{\mathfrak{S}} (\vec{\boldsymbol{q}}_{\boldsymbol{\ell}}, [y_\ell z_\ell]) = (\vec{\boldsymbol{r}}, [w])$$

which is a run omitting the configuration $(\vec{\boldsymbol{q}}, [v]) = (\vec{\boldsymbol{q}}_{\boldsymbol{0}}, [c_0 z_0])$. Note that the first transition of this run replaces a by x' which is shorter than x. Hence, the induction hypothesis yields an action sequence $s \in \prod_{k=2}^{n+1} \overline{A}^* (\overline{A}_{=a_k} A^+)^+$ with $(\vec{\boldsymbol{p}}, [u]) \xrightarrow{s}_{\mathfrak{S}} (\vec{\boldsymbol{r}}, [w])$.

Now, suppose $y_k \neq \varepsilon$. Since this transition writes at least one letter, there must be an $1 \leq m \leq n+1$ with $c_k\,\mathrm{M} = a_m\,\mathrm{M}$. If we have $m \leq n$, we get

$$a_m\,\mathrm{M} = c_k\,\mathrm{M} = b\,\mathrm{M} \subseteq a_0\,\mathrm{M} \subseteq a_{n+1}\,\mathrm{M}\,,$$

i.e., $a_m \leq a_{n+1}$ which contradicts $a_m \not\leq a_{n+1}$. Hence, we have $m = n+1$, i.e.,

$$b\,\mathrm{M} \subseteq a_0\,\mathrm{M} \subseteq a_{n+1}\,\mathrm{M} = b\,\mathrm{M}\,.$$

In other words, we have $b, a_0 \in A_{=a_{n+1}}$. Since $b \parallel c_j$ for each $0 \leq j < k$ we can again reorder our transitions as follows:

- first apply the *local* transitions $(\vec{\boldsymbol{q}}_{\boldsymbol{j}}{\restriction_{c_j\,\mathrm{M}}}, \overline{c_j} y_j, \vec{\boldsymbol{c}}_{\boldsymbol{j+1}}\,\mathrm{M}\,{\restriction_{c_j\,\mathrm{M}}})$ for each $0 \leq j < k$,
- then apply the *local* transition $(\vec{\boldsymbol{p}}{\restriction_{a_0\,\mathrm{M}}}, \overline{a_0} x, \vec{\boldsymbol{q}}{\restriction_{a_0\,\mathrm{M}}})$, and
- finally follow the original run from $(\vec{\boldsymbol{q}}_{\boldsymbol{k}}, [c_k z_k])$ to $(\vec{\boldsymbol{r}}, [w])$.

Note that the transitions of the third part of this run always read letters from $A_{=a_{n+1}}$. Hence, we infer

$$s := \overline{c_0} y_0 \overline{c_1} y_1 \ldots \overline{c_{k-1}} y_{k-1}\, \overline{a_0} x\, \overline{c_k} y_k \ldots \overline{c_\ell} y_\ell \in \prod_{k=2}^{n+1} \overline{A}^* (\overline{A}_{=a_k} A^+)^+$$

and $(\vec{\boldsymbol{p}}, [u]) \xrightarrow{s}_{\mathfrak{S}} (\vec{\boldsymbol{r}}, [w])$ which proves our claim for the second case. ◁

Finally, by iterated application of our claim we can reduce the number of transitions in the first increase phase to 0. Since R is idempotent, we infer the statement of this lemma. ◂

From Lemma 9.5.27 we infer the existence of a bound $\hat{n} \in \mathbb{N}$ such that for each $w \in A^{>\hat{n}}$ there is a word $f(w) \in A^{\leq \hat{n}}$ with $\mathrm{RW}_{=w}(D) \subseteq \mathrm{RW}_{=f(w)}(D)$ for any set $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$ of configurations. This finally implies

$$\mathrm{post}^*_{\mathfrak{S}}(D) = (\mathrm{RW})^*(D) = \bigcup_{w \in A^{\leq \hat{n}}} \mathrm{RW}_{=w}(D)\,.$$

Hence, if D is rational, then $\mathrm{post}^*_{\mathfrak{S}}(D)$ is a finite union of rational sets implying the rationality of $\mathrm{post}^*_{\mathfrak{S}}(D)$. The following proposition summarizes this observation and shows that the bound $\hat{n}$ equals $|A| + 1$.

Proposition 9.5.28. *Let $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$ be a set of configurations. Then we have*

$$(\mathrm{RW})^*(D) = (\mathrm{RW})^{|A|+1}(D)\,.$$

Proof. The inclusion "$\supseteq$" is obvious. So, we only have to show the converse inclusion "$\subseteq$". To this end, let $c \in (\mathrm{RW})^*(D) = \bigcup_{w \in A^*} \mathrm{RW}_{=w}\,\mathrm{R}(D)$. Hence, there is a word $w \in A^*$ with $c \in \mathrm{RW}_{=w}\,\mathrm{R}(D)$. We choose such word $w \in A^*$ with minimal length. If $|w| \leq |A|$ holds, we are done. So, assume $|w| > |A|$. Let $w = a_1 a_2 \dots a_k$ with $a_1, a_2, \dots, a_k \in A$ and $k > |A|$.

Since the number of equivalence classes induced by the quasi ordering $\leq$ is at most $|A|$, there are two indices $1 \leq m < n \leq k$ with $a_m \leq a_n$ according to the pigeon hole principle. We choose m and n with this property such that the distance $n - m$ is minimal. Hence, we obtain $a_{m+1}, a_{m+2}, \dots, a_{n-1} \not\leq a_n$. Then by Lemma 9.5.27 we learn

$$c \in \mathrm{RW}_{=w}\,\mathrm{R}(D) = \mathrm{RW}_{=a_1 \dots a_{m-1} a_m a_{m+1} \dots a_n \dots a_k}\,\mathrm{R}(D) = \mathrm{RW}_{=a_1 \dots a_{m-1} a_{m+1} \dots a_n \dots a_k}\,\mathrm{R}(D)\,.$$

However, the word $w' := a_1 \dots a_{m-1} a_{m+1} \dots a_n \dots a_k$ is shorter than w and satisfies the property $c \in \mathrm{RW}_{w'}\,\mathrm{R}(D)$. This is a contradiction to the choice of w being a word of minimal length. ◂

According to Proposition 9.5.28 we find for any run of the saturated system $\mathfrak{S}$ another run, which alternates at most $|A| + 1$ times between increase and decrease phases. From the proof of this statement we obtain a possibly tighter upper bound for the number of alternations: $N + 1$ where N is the number of equivalence classes induced by the quasi-ordering $\leq$. For example, if $A = \{a, b\} = A_{=a}$ (i.e., $\mathfrak{S}$ is a single-pushdown system), Proposition 9.5.28 yields 3 alternations while we have only one equivalence class implying that 2 is a tighter bound (note that according to [FWW97] 1 is also an upper bound in this case).

Finally, we can prove Theorem 9.5.16 by bringing together all of the previously described steps:

Proof (of Theorem 9.5.16). From $\mathfrak{S}$ we compute the asynchronous pushdown system $\mathfrak{S}^{(\infty)}$ by adding shortcuts as described in Step 1. Then we know $\mathrm{post}^*_{\mathfrak{S}}(C) = \mathrm{post}^*_{\mathfrak{S}^{(\infty)}}(C)$ by Lemma 9.5.19. From Propositions 9.5.25 and 9.5.28 we learn that

$$\mathrm{post}^*_{\mathfrak{S}}(C) = \mathrm{post}^*_{\mathfrak{S}^{(\infty)}}(C) = (\mathrm{RW})^*(C) = (\mathrm{RW})^{|A|+1}(C)$$

holds. Finally, iterated application of Lemmata 9.5.23 and 9.5.24 yields an $\mathfrak{S}$-NFA accepting $(\mathrm{RW})^{|A|+1}(C) = \mathrm{post}^*_{\mathfrak{S}}(C)$. If we assume that $\mathcal{A}$ is fixed, this automaton can be computed in time polynomial in the size of $\mathfrak{S}$ and an $\mathfrak{S}$-NFA accepting C. ◂

Now, let $\mathcal{A}$ be a distributed alphabet and $\mathfrak{S}$ be an asynchronous pushdown system over $\mathcal{A}$. A relation $R \subseteq \mathrm{Conf}_{\mathfrak{S}} \times \mathrm{Conf}_{\mathfrak{S}}$ is called *rational* if for each pair of control states $\vec{p}$ and $\vec{q}$ of $\mathfrak{S}$ the relation

$$\{(\lambda, \kappa) \mid (\vec{p}, \lambda)\, R\, (\vec{q}, \kappa)\} \tag{9.5}$$

is rational in $\mathbb{M}(\mathcal{A})^2$. Note that the relation in (9.5) is a projection to the distributed stack's contents of the configurations in R with control states $\vec{p}$ and $\vec{q}$.

As a corollary of Theorem 9.5.16 we infer that the relation of all configurations $c, d \in \mathrm{Conf}_{\mathfrak{S}}$ of $\mathfrak{S}$ satisfying $c \to_{\mathfrak{S}}^* d$ is efficiently rational:

Theorem 9.5.29. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\mathfrak{S} = (\vec{Q}, \mathcal{P}_{\mathcal{A}}, \Delta, \Lambda)$ be an asynchronous pushdown system. Then the relation*

$$\{(c, d) \in \mathrm{Conf}_{\mathfrak{S}} \times \mathrm{Conf}_{\mathfrak{S}} \mid c \to_{\mathfrak{S}}^* d\}$$

is rational. In particular, from $\mathfrak{S}$ we can compute a representation of this relation. If $\mathcal{A}$ is fixed, this construction is possible in polynomial time.

Proof. Before we show the correctness of this theorem we want to sketch the idea of the proof. To this end, recall that tuples of traces from $\mathbb{M}(\mathcal{A})$ are essentially traces from another trace monoid $\mathbb{M}(\mathcal{A}')$ where $\mathcal{A}'$ consists of two independent copies of $\mathcal{A}$. In other words, we can see the relation $R := \{(c, d) \mid c \to_{\mathfrak{S}}^* d\}$ as a set of configurations of another asynchronous pushdown system $\mathfrak{S}'$ over the modified distributed alphabet $\mathcal{A}'$. This system simulates $\mathfrak{S}$ on one component of the tuples from R and leaves the other one untouched. Then starting from the identity relation on $\mathrm{Conf}_{\mathfrak{S}}$ (this is a rational set of configurations of $\mathfrak{S}'$) the set of forwards reachable configurations in $\mathfrak{S}'$ is R. Finally, we can compute an $\mathfrak{S}'$-NFA accepting R according to Theorem 9.5.16.

First, our modified distributed alphabet $\mathcal{A}' = (A', P', \mathrm{M}')$ is the following one:

- $A' := \hat{A} \cup A$ where $\hat{A} := \{\hat{a} \mid a \in A\}$ is a disjoint copy of A,
- $P' := \hat{P} \cup P$ where $\hat{P} := \{\hat{i} \mid i \in P\}$ is a disjoint copy of P, and
- $\mathrm{M}' := \{(\hat{a}, \hat{i}) \mid a \,\mathrm{M}\, i\} \cup \mathrm{M}$.

Then we construct an asynchronous pushdown system $\mathfrak{S}' = (\vec{Q}', \mathcal{P}_{\mathcal{A}'}, \Delta', \Lambda')$ over the modified distributed alphabet $\mathcal{A}'$ as follows:

- $\vec{Q}' := \vec{Q} \times \vec{Q}$ (we assume that the first copy of $\vec{Q}$ represents the processes $\hat{P}$ and the second one represents P),
- $\Delta' := \{((\vec{p}_1, \vec{p}_2), \overline{a}t, (\vec{p}_1, \vec{q})) \mid (\vec{p}_2, \overline{a}t, \vec{q}) \in \Delta\}$, and
- λ' is arbitrary.

Note that $\mathfrak{S}'$ is asynchronous since we only simulate the transitions of $\mathfrak{S}$ and do not touch the processes from $\hat{P}$. Finally, we set

$$C := \{(\vec{q}, \vec{q}) \mid \vec{q} \in \vec{Q}\} \times \{[\hat{a}a] \mid a \in A\}^* \subseteq \mathrm{Conf}_{\mathfrak{S}'}.$$

In other words, the set of initial configurations is essentially the identity relation on $\mathrm{Conf}_{\mathfrak{S}}$. Note that this set of configurations of $\mathfrak{S}'$ is efficiently rational.

By Theorem 9.5.16 we can construct an $\mathfrak{S}'$-NFA $\mathfrak{A}$ accepting the set $\mathrm{post}_{\mathfrak{S}'}^*(C)$ in time polynomial in the size of $\mathfrak{S}'$ and an $\mathfrak{S}'$-NFA accepting C. Then for $\vec{p}, \vec{q} \in \vec{Q}$ the set

$$L_{\vec{p},\vec{q}} := \{\lambda \in \mathbb{M}(\mathcal{A}') \mid \{((\vec{p}, \vec{q}), \lambda)\} \in C(\mathfrak{A})\}$$

is rational in $\mathbb{M}(\mathcal{A}')$. Note that this set contains exactly those traces $\lambda \in \mathbb{M}(\mathcal{A}')$ satisfying $(\vec{p}, \phi(\lambda)) \to_{\mathfrak{S}}^* (\vec{q}, \pi_A(\lambda))$ where $\phi\colon \mathbb{M}(\mathcal{A}') \to \mathbb{M}(\mathcal{A})$ first projects a trace to the alphabet $\hat{A}$ and afterwards suppresses the hats from each letter.

Finally, using the homomorphism $\psi\colon \mathbb{M}(\mathcal{A}') \to \mathbb{M}(\mathcal{A})^2$ induced by $\psi(a) = (\varepsilon, a)$ and $\psi(\hat{a}) = (a, \varepsilon)$ for each $a \in A$ (note that this homomorphism is well-defined), we obtain the efficient rationality of

$$\psi(L_{\vec{p},\vec{q}}) = \{(\kappa, \lambda) \mid (\vec{p}, \kappa) \to_{\mathfrak{S}}^* (\vec{q}, \lambda)\}$$

implying that $\{(c, d) \in \mathrm{Conf}_{\mathfrak{S}} \times \mathrm{Conf}_{\mathfrak{S}} \mid c \to_{\mathfrak{S}}^* d\}$ is efficiently rational. ◂

Note that it is decidable whether a given rational trace language is contained in a given recognizable trace language. Hence, our results, Theorems 9.5.9 and 9.5.16, allow to decide the following problems for a given asynchronous pushdown system $\mathfrak{S}$, a rational set $C \subseteq \mathrm{Conf}_{\mathfrak{S}}$ of configurations, and a recognizable set $D \subseteq \mathrm{Conf}_{\mathfrak{S}}$ of configurations:

(1) *Reachability Problem.* Can we reach D from C, or equivalently, does $\mathrm{post}_{\mathfrak{S}}^*(C) \cap D \neq \varnothing$ (resp. $C \cap \mathrm{pre}_{\mathfrak{S}}^*(D) \neq \varnothing$) hold?

(2) *Safety Problem.* Can we only reach D from C, or equivalently, does $\mathrm{post}_{\mathfrak{S}}^*(C) \subseteq D$ hold?

(3) *Liveness Problem.* Can we reach D from any reachable configuration of C, or equivalently, does $\mathrm{post}_{\mathfrak{S}}^*(C) \subseteq \mathrm{pre}_{\mathfrak{S}}^*(D)$ hold?

Note that until now we cannot decide the *Inevitability Problem* (do we stay in C until we eventually reach D?). To solve this problem we should first decide the *Recurrent Reachability Problem* of asynchronous pushdown systems.

9.5.3 Recurrent Reachability

Towards the recurrent reachability problem we should recall that this problem is equivalent to asking whether an asynchronous Büchi-pushdown automaton accepts at least one infinite word. With the help of Theorem 9.5.9 we will see that this problem is decidable in polynomial time. We will use the recurrent reachability problem to prove the decidability of special temporal logics. Concretely, we will obtain that the model checking problem of an LTL-like logic for traces is decidable in exponential time for Kripke-structures induced by asynchronous pushdown systems.

Lemma 9.5.30. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and let $\mathfrak{A} = (\vec{Q}, \Gamma, \mathcal{P}_{\mathcal{A}}, I, c, \Delta, F)$ be an asynchronous pushdown automaton (without ε-transitions). Then we have $L^\omega(\mathfrak{A}) \neq \varnothing$ if, and only if, there are $\vec{q} \in \vec{Q}$ and $a \in A$ with*

(1) $\mathrm{pre}_{\mathfrak{A}}^*(\{\vec{q}\} \times ([a] \cdot \mathbb{M}(\mathcal{A}))) \cap \mathrm{Init}_{\mathfrak{A}} \neq \varnothing$ *and*

(2) $(\vec{q}, [a]) \in \mathrm{pre}_{\mathfrak{A}}^+\Big(\mathrm{Final}_{\mathfrak{A}} \cap \mathrm{pre}_{\mathfrak{A}}^+\big(\{\vec{q}\} \times ([a] \cdot \mathbb{M}(\mathcal{A}))\big)\Big)$.

Proof. First, we assume that (1) and (2) hold. Then, there are $u \in \Gamma^*$, $v, w \in \Gamma^+$, $x, y, z \in A^*$, $(\vec{\imath}, c) \in \mathrm{Init}_{\mathfrak{A}}$, and $(\vec{f}, [x]) \in \mathrm{Final}_{\mathfrak{A}}$ with

- $(\vec{\boldsymbol{\iota}}, c) \xrightarrow{u}_{\mathfrak{A}} (\vec{\boldsymbol{q}}, [az])$ and
- $(\vec{\boldsymbol{q}}, [a]) \xrightarrow{v}_{\mathfrak{A}} (\vec{\boldsymbol{f}}, [x]) \xrightarrow{w}_{\mathfrak{A}} (\vec{\boldsymbol{q}}, [ay])$.

From these properties we obtain the following infinite run:

$$(\vec{\boldsymbol{\iota}}, c) \xrightarrow{u}_{\mathfrak{A}} (\vec{\boldsymbol{q}}, [az]) \xrightarrow{v}_{\mathfrak{A}} (\vec{\boldsymbol{f}}, [xz]) \xrightarrow{w}_{\mathfrak{A}} (\vec{\boldsymbol{q}}, [ayz]) \xrightarrow{v}_{\mathfrak{A}} (\vec{\boldsymbol{f}}, [xyz]) \xrightarrow{w}_{\mathfrak{A}} (\vec{\boldsymbol{q}}, [ay^2z]) \xrightarrow{v}_{\mathfrak{A}} \dots$$

This run visits infinitely often $\vec{\boldsymbol{f}} \in F$, i.e., we have $u(vw)^\omega \in L^\omega(\mathfrak{A}) \neq \varnothing$.

Conversely, we assume $L^\omega(\mathfrak{A}) \neq \varnothing$. Then there are $\alpha_0, \alpha_1, \dots \in \Gamma$, $\vec{\boldsymbol{q}}_{\boldsymbol{0}}, \vec{\boldsymbol{q}}_{\boldsymbol{1}}, \dots \in \vec{\boldsymbol{Q}}$, and $x_0, x_1, \dots \in A^*$ with $\alpha_0\alpha_1\alpha_2\dots \in L^\omega(\mathfrak{A})$, $(\vec{\boldsymbol{q}}_{\boldsymbol{0}}, [x_0]) \in \mathrm{Init}_{\mathfrak{A}}$,

$$(\vec{\boldsymbol{q}}_{\boldsymbol{0}}, [x_0]) \xrightarrow{\alpha_0}_{\mathfrak{A}} (\vec{\boldsymbol{q}}_{\boldsymbol{1}}, [x_1]) \xrightarrow{\alpha_1}_{\mathfrak{A}} (\vec{\boldsymbol{q}}_{\boldsymbol{2}}, [x_2]) \xrightarrow{\alpha_2}_{\mathfrak{A}} \dots,$$

and infinitely many $j \in \mathbb{N}$ with $(\vec{\boldsymbol{q}}_{\boldsymbol{j}}, [x_j]) \in \mathrm{Final}_{\mathfrak{A}}$. Since this is a run of $\mathfrak{A}$, there are transitions $(\vec{\boldsymbol{q}}_{\boldsymbol{j}}, \alpha_j, \overline{a_j}t_j, \vec{\boldsymbol{q}}_{\boldsymbol{j+1}}) \in \Delta$ with $[\![\overline{a_j}t_j]\!]([x_j]) = [x_{j+1}]$ for each $j \in \mathbb{N}$.

▷ *Idea.* Since the run $(\vec{\boldsymbol{q}}_{\boldsymbol{k}}, [x_k])_{k\in\mathbb{N}}$ does not necessarily satisfy the two conditions of our lemma, we will construct another accepting run of $\mathfrak{A}$ which actually satisfies these conditions. To this end, we want to find a letter $a \in A$ which is read infinitely often from the stacks. We choose this letter maximal with this property wrt. the well-quasi-ordering $\leq$. We show then that each infinitely often occurring letter is either independent or smaller than a wrt. $\leq$.

Now, we obtain our new run from the original one as follows: both runs agree in their (sufficiently long) beginning. Afterwards, we remove all transitions reading letters independent of a. So, after this long beginning the run acts like a single-pushdown system on the processes $a\,\mathrm{M}$. Hence, the rest of this proof will generalize the idea known from [BEM97]. ⌟

For $k \in \mathbb{N}$ we set $A_k := \{a_j \mid j \geq k\} \subseteq A$, i.e., A_k is the set of all letters we read on our infinite run starting from the k^{th} step. By definition we have $A \supseteq A_0 \supseteq A_1 \supseteq \dots$, i.e., the sequence $(A_k)_{k\in\mathbb{N}}$ is monotonically decreasing. Since $\subseteq$ is well-quasi-ordered on finite sets, there is an integer $k \in \mathbb{N}$ such that $A_k = A_\ell$ for each $\ell \geq k$ holds. It is easy to see that each letter $a \in A_k$ is read infinitely often on our run. Note that our run also visits infinitely often an accepting state after the k^{th} step. In particular, by the pigeon hole principle there is a state $\vec{\boldsymbol{f}} \in F$ which appears infinitely often on our run. W.l.o.g., we can assume that $\vec{\boldsymbol{q}}_{\boldsymbol{k}} = \vec{\boldsymbol{f}}$ holds.

Recall the quasi-ordering $\leq$ on A with $a \leq b$ iff $a\,\mathrm{M} \subseteq b\,\mathrm{M}$. Since A is finite, $\leq$ is even a well-quasi-ordering. We choose a letter $a \in A_k$ which is maximal wrt. $\leq$. Then we can show that each letter $b \in A_k$ is either smaller than a wrt. $\leq$ or is independent of a.

▷ *Claim 1. Let $b \in A_k$. Then we have either $b \leq a$ or $b \parallel a$.*

Proof idea. Towards a contradiction, assume there is $b \in A_k$ with $b \not\leq a$ and $b \nparallel a$. W.l.o.g., b also is maximal wrt. $\leq$. Let $i \in a\,\mathrm{M} \cap b\,\mathrm{M}$ be a common process. Note that due to maximality of a and b we have $a\,\mathrm{M} \setminus b\,\mathrm{M} \neq \varnothing$ and $b\,\mathrm{M} \setminus a\,\mathrm{M} \neq \varnothing$, i.e., by asynchronism of $\mathfrak{A}$ it is impossible that we read a (resp., b) and write b (resp., a) in the same transition.

Now let $j_1 \geq k$ with $\pi_i(x_{j_1}) \in aA_i^*$ (this index j_1 exists since we read $a \in A_k$ infinitely often). Since we have $b \in A_k$ there is another computation step $k_2 > j_1$ in which we read b, i.e., we have $\pi_i(x_{k_2}) \in bA_i^*$.

Then, in order to reach the content $\pi_i(x_{k_2})$ after step j_1, we first have to remove the letter a and any other letter from $A_{\leq a} \cap A_i$ which we have pushed onto stack i between the j_1^{th} and k_2^{th} step. This is due to the restriction of asynchronous pushdown automata writing at most

letters onto stacks from which we have read a letter before (cf. Observation 9.5.5). In other words, there is a step $j_1 < i_2 \leq k_2$ with $\pi_i(x_{j_1}) = a\pi_i(x_{i_2})$.

Since we have the letter $b \in A_k$ on top of stack i in the k_2^{th} computation step, there was a letter $c \in A_{=b}$ in stack i's content at step i_2 which is either exactly this b we read in step k_2 or which (transitively) writes b onto stack i. This means, we have $\pi_i(x_{i_2}) \in A_i^* c A_i^*$ and $\pi_i(x_{j_1}) \in a A_i^* c A_i^*$. Hence, there is a computation step $i_2 \leq j_2 \leq k_2$ with $\pi_i(x_{j_2}) \in c A_i^*$ and, therefore

$$|\pi_i(x_{j_1})| > |\pi_i(x_{i_2})| \geq |\pi_i(x_{j_2})|.$$

By exchanging the roles of a and b we similarly find computation steps $k_2 < i_3 \leq j_3 \leq k_3$ with the following properties:

- $\pi_i(x_{j_2}) = a\pi_i(x_{i_3})$,
- $\pi_i(x_{k_3}) \in cA_i^*$, and
- $\pi_i(x_{j_3}) \in dA_i^*$ for a letter $d \in A_{=a}$ with $\pi_i(x_{j_2}) \in cA_i^* dA_i^*$.

We could similarly infer $|\pi_i(x_{j_2})| < |\pi_i(x_{j_3})|$.

By induction we finally obtain an infinite sequence $j_1 < j_2 < j_3 < j_4 < \dots$ with

$$|\pi_i(x_{j_1})| > |\pi_i(x_{j_2})| > |\pi_i(x_{j_3})| > |\pi_i(x_{j_4})| > \dots,$$

i.e., stack i's height is monotonically decreasing. However, this is impossible since $(\mathbb{N}, \leq)$ is a well-quasi-ordering. ◁

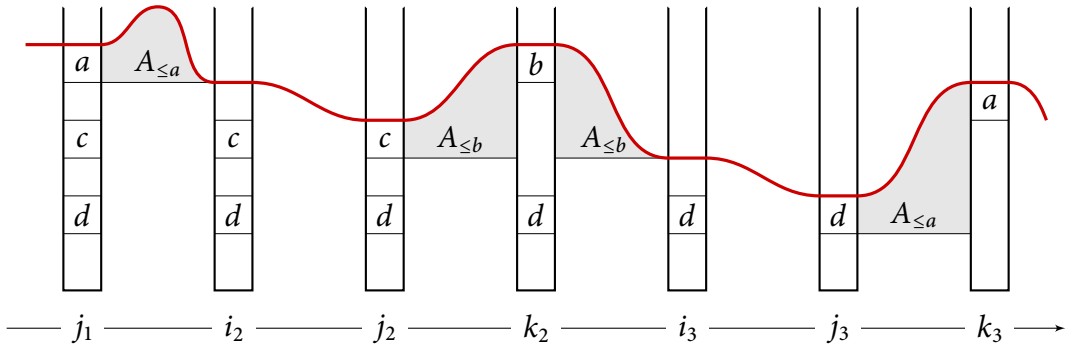

Figure 9.15. The proof of Claim 1. We visualize the contents of stack i over time. In this connection, the red (thick) line marks the height of stack i. The gray shaded areas mark the letters (transitively) written by a, c, resp. d.

Now we want to construct another infinite run of $\mathfrak{A}$ which only reads letters from $A_{\leq a}$ after the k^{th} computation step. We do this by deletion of all other transitions.

▷ *Claim 2. There are $\vec{\boldsymbol{p}}_0, \vec{\boldsymbol{p}}_1, \vec{\boldsymbol{p}}_2, \dots \in \vec{\mathbf{Q}}$, $y_0, y_1, y_2, \dots \in A^*$, and $\beta_0, \beta_1, \dots \in \Gamma$ such that the following properties hold:*

(i) $(\vec{\boldsymbol{q}}_k, [x_k]) = (\vec{\boldsymbol{p}}_0, [y_0])$,

(ii) for each $j \in \mathbb{N}$ there is $b \in A_{\leq a} \cap A_k$ and $t \in A^*$ such that $(\vec{\boldsymbol{p}}_j, \beta_j, \overline{bt}, \vec{\boldsymbol{p}}_{j+1}) \in \Delta$ and $[\![\overline{bt}]\!]([y_j]) = [y_{j+1}]$, and

(iii) there are infinitely many $j \in \mathbb{N}$ with $(\vec{\boldsymbol{p}}_j, [y_j]) \in \text{Final}_{\mathfrak{A}}$.

Proof. We inductively construct global states $\vec{q}'_k, \vec{q}'_{k+1}, \dots \in \vec{Q}$, stack contents $x'_k, x'_{k+1}, \dots \in A^*$, and letters $\alpha'_k, \alpha'_{k+1}, \dots \in \Gamma \cup \{\varepsilon\}$ as follows: we set $\vec{q}'_k := \vec{q}_k$ and $x'_k := x_k$. Now let $j \geq k$. Then we distinguish two cases:

- if $a_j \notin A_{\leq a}$ we set $\alpha'_j := \varepsilon$, $\vec{q}'_{j+1} := \vec{q}'_j$, and $x'_{j+1} := x'_j$.
- if $a_j \in A_{\leq a}$ we set $\alpha'_j := \alpha_j$, $\vec{q}'_{j+1} := (\vec{q}_{j+1}{\restriction}_{a\,\mathrm{M}}, \vec{q}_k{\restriction}_{P \setminus a\,\mathrm{M}})$ and $x'_{j+1} \in A^*$ with $[x'_{j+1}] = [\![\overline{a_j} t_j]\!]([x'_j])$. Note that x'_{j+1} arises from x_{j+1} and x_k by combining the stacks in $a\,\mathrm{M}$ of x_{j+1} with the stacks in $P \setminus a\,\mathrm{M}$ of x_k. Hence, x'_{j+1} is well-defined.

In other words, whenever we read a letter $b \in A$ which is independent from a in the original run, we remove the effect of this step in our new run. Whenever we read a letter $b \in A_{\leq a}$, we also apply this transition in our new run. Note that this is possible due to the asynchronism of $\mathfrak{A}$. Since $\vec{q}'_k = \vec{q}_k = \vec{f} \in F$ holds and since $\vec{f}$ occurs infinitely often in our original run, $\vec{f}$ also occurs infinitely often in our new run.

Finally, removing $\vec{q}'_{j+1}$, x'_{j+1}, and α'_j from these sequences whenever $a_j \notin A_{\leq a}$ holds and re-indexing yields the sought run. ◁

Next, we consider a subsequence of our new run where the height of our stacks is not decreasing. To this end, we set $I := \{j \geq k \mid \forall \ell \geq j\colon |y_j| \leq |y_\ell|\}$. Since the natural ordering on $\mathbb{N}$ is a well-ordering, this sequence is infinite. Using the pigeon hole principle we can restrict this subsequence to those configurations having the same control state and the same top of our stacks. Formally, there are $\vec{p} \in \vec{Q}$ and $b \in A_k \cap A_{\leq a}$ such that $J := \{j \in I \mid \vec{p} = \vec{p}_j, [y_j] \in [b] \cdot \mathbb{M}(\mathcal{A})\}$ is infinite. We can also assume that $b \in A_{=a}$ holds, i.e., b is on top of all considered stacks $i \in a\,\mathrm{M}$ in infinitely many configurations with minimal stack heights. Finally, we can prove that (1) and (2) hold:

Let $j \in J$. Then by definition we have $(\vec{q}_0, [x_0]) \in \mathrm{pre}^*_{\mathfrak{A}}(\{(\vec{p}_j, [y_j])\}) \cap \mathrm{Init}_{\mathfrak{A}}$ (recall that $[y_j] \in [b] \cdot \mathbb{M}(\mathcal{A})$ holds). Towards the second property, let $\ell > j$ with $\vec{p}_\ell = \vec{f} \in F$. This number exists, since we visit $\vec{f}$ infinitely often. Additionally, since J is infinite, there is $m \in J$ with $m > \ell > j$, i.e., we have $\vec{p}_m = \vec{p}$ and $[y_m] \in [b] \cdot \mathbb{M}(\mathcal{A})$. Since $j \in J \subseteq I$ we have $|y_j| \leq |y_n|$ for each $n \geq j$. Since $b \in A_{=a}$ affects all processes from $a\,\mathrm{M}$ and since we do not touch the other processes anymore, we also have $|y_j|_{A_i} \leq |y_n|_{A_i}$ for each $n \geq j$ and $i \in P$. Hence, for $[y_j] = [bz]$ (where $z \in A^*$) we also have $[y_n] \in \mathbb{M}(\mathcal{A}) \cdot [z]$. This finally implies

$$(\vec{p}, [b]) \in \mathrm{pre}^+_{\mathfrak{A}}\big(\mathrm{Final}_{\mathfrak{A}} \cap \mathrm{pre}^+_{\mathfrak{A}}(\{\vec{p}\} \times ([b] \cdot \mathbb{M}(\mathcal{A})))\big)\,. \qquad \blacktriangleleft$$

From Lemma 9.5.30 we learn that the recurrent reachability problem resp. the emptiness problem of asynchronous Büchi-pushdown automata is decidable. However, we cannot apply the reduction from Theorem 8.3.1 to solve the model checking problem of LTL for the Kripke-structures induced by asynchronous pushdown systems. This is due to the fact that the constructed Büchi-automaton $\mathfrak{A}$ for an LTL-formula ϕ (as described in the previous chapter) is not necessarily asynchronous. Therefore, the Büchi-pushdown automaton $\mathfrak{B}$ simulating $\mathfrak{A}$ and the given asynchronous pushdown system $\mathfrak{S}$ in parallel is not necessarily asynchronous.

We can circumvent this problem with a more specialized version of LTL. Concretely, we replace the temporal operators X and U by local ones. So, we introduce two local operators X_i and U_i for any process $i \in P$. In this connection, "$\mathrm{X}_i\,\phi$" means "ϕ holds on the next step of process i" and "$\phi\,\mathrm{U}_i\,\psi$" means "ϕ holds on process i until ψ holds". This is essentially the logic

TrPTL (PTL resp. LTL for traces) introduced by Thiagaranjan in [Thi94]. We also require that the labeling Λ of an asynchronous pushdown system $\mathfrak{S} = (\vec{\boldsymbol{Q}}, \mathcal{P}_{\mathcal{A}}, \Delta, \Lambda)$ consists of local labelings Λ_i. In other words, we have pairwise disjoint sets AP_i of atomic propositions and functions $\Lambda_i \colon Q_i \to 2^{\mathrm{AP}_i}$ such that $\Lambda((q_i)_{i\in P}) = \bigcup_{i\in P} \Lambda_i(q_i)$ for each $i \in P$ holds. In this case, we call $\mathfrak{S}$ an asynchronous pushdown system *with local labeling*.

Now consider the formula $\phi = \mathrm{G}_i \mathrm{F}_i\, a \wedge \mathrm{G}_j \mathrm{F}_j\, b$ where $a \parallel b$, $a \,\mathrm{M}\, i$, and $b \,\mathrm{M}\, j$ holds. Then ϕ states that on a run of a given asynchronous pushdown system $\mathfrak{S}$ (with local labeling) we read a and b infinitely often. In other words, ϕ states a special fairness property of the processes i and j. Unfortunately, we cannot check whether ϕ holds using Lemma 9.5.30. This is due to the fact that our proof of Lemma 9.5.30 considers runs only affecting a connected set of processes of $\mathfrak{S}$.

Hence, towards the proof strategy sketched in Section 8.3 we also require a local accepting condition of asynchronous Büchi-pushdown automata. Such conditions were introduced by Muscholl in [Mus94]: concretely, we require the Büchi-automata on traces to visit all states of a given set of local states infinitely often for any process $i \in P$. To this end, we first have to determine the local states an automaton visits infinitely often on a given run:

Let $\mathfrak{A} = (\vec{\boldsymbol{Q}}, \Gamma, \mathcal{P}_{\mathcal{A}}, I, c, \Delta, F)$ be an asynchronous pushdown automaton with $\vec{\boldsymbol{Q}} = \prod_{i\in P} Q_i$ and $\rho = ((q_{i,j})_{i\in P}, [w_j])_{j\in\mathbb{N}}$ be an infinite run of $\mathfrak{A}$ reading the letters $a_1, a_2, \dots \in A$ from its distributed stack. For a process $i \in P$ we denote the set of all local states from Q_i occurring infinitely often in ρ by $\inf_i(\rho) := \{q \in Q_i \mid \exists^\infty j \in \mathbb{N} \colon q_{i,j} = q\}$ if there are infinitely many $j \in \mathbb{N}$ with $a_j \,\mathrm{M}\, i$. Otherwise we set $\inf_i(\rho) := \varnothing$ (in this case the run on process i is finite). A *table* is a set $\mathcal{T} \subseteq \prod_{i\in P} 2^{Q_i}$. Then the accepted ω-language of $\mathfrak{A}$ wrt. a table $\mathcal{T}$ is the following one:

$$L^{\omega}_{\mathcal{T}}(\mathfrak{A}) := \left\{ \alpha_0\alpha_1 \dots \in \Gamma^{\omega} \;\middle|\; \begin{array}{c} \exists \text{ infinite run } \rho \text{ labeled with } \alpha_0\alpha_1 \dots \text{ in } \mathfrak{A}, (T_i)_{i\in P} \in \mathcal{T} \colon \\ \forall\, i \in P \colon T_i \subseteq \inf_i(\rho) \end{array} \right\}.$$

Next, we want to prove the decidability of the emptiness problem of ω-languages accepted by asynchronous Büchi-pushdown automata with local accepting conditions. We will see that the proof of this statement is close to the proof of Lemma 9.5.30, but a bit more involved. We should also note that we only consider tables $\mathcal{T}$ of cardinality 1. However, we obtain the general case easily by splitting $\mathcal{T}$ into $|\mathcal{T}|$ singletons.

Lemma 9.5.31. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\mathfrak{A} = (\vec{\boldsymbol{Q}}, \Gamma, \mathcal{P}_{\mathcal{A}}, I, c, \Delta, F)$ be an asynchronous pushdown automaton (without ε-transitions), and let $T_i = \{f_{i,1}, \dots, f_{i,n_i}\} \subseteq Q_i$ for each $i \in P$. Set $\mathcal{T} = \{(T_i)_{i\in P}\}$. Then we have $L^{\omega}_{\mathcal{T}}(\mathfrak{A}) \neq \varnothing$ if, and only if, there are $\vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}$ and $w \in A^{\leq |P|}$ with*

(1) $\mathrm{pre}^*_{\mathfrak{A}}\big(\{\vec{\boldsymbol{q}}\} \times ([w] \cdot \mathbb{M}(\mathcal{A}))\big) \cap \mathrm{Init}_{\mathfrak{A}} \neq \varnothing$ *and*

(2) for each $i \in P$ and $1 \leq j \leq n_i$ we have

$$(\vec{\boldsymbol{q}}, [w]) \in \mathrm{pre}^+_{\mathfrak{A}}\big((\vec{\boldsymbol{Q}}{\restriction}_{P\setminus\{i\}}, \{f_{i,j}\}) \times \mathbb{M}(\mathcal{A}) \cap \mathrm{pre}^+_{\mathfrak{A}}(\{\vec{\boldsymbol{q}}\} \times ([w] \cdot \mathbb{M}(\mathcal{A})))\big).$$

Proof. This proof is similar to the proof of Lemma 9.5.30 with some generalizations.

Again, we first assume that the properties (1) and (2) hold. Then there are (finite) words $u \in \Gamma^*, v_{i,j}, w_{i,j} \in \Gamma^+$ and $x, y_{i,j}, z_{i,j} \in A^*$, $\vec{\boldsymbol{\imath}} \in I$, and $\vec{\boldsymbol{p}}_{i,j} \in (\vec{\boldsymbol{Q}}\restriction_{P \setminus \{i\}}, \{f_{i,j}\})$ (for each $i \in P$ and $1 \leq j \leq n_i$) with

- $(\vec{\boldsymbol{\imath}}, c) \xrightarrow{u}_{\mathfrak{A}} (\vec{\boldsymbol{q}}, [wx])$ and
- $(\vec{\boldsymbol{q}}, [wx]) \xrightarrow{v_{i,j}}_{\mathfrak{A}} (\vec{\boldsymbol{p}}_{i,j}, [y_{i,j}]) \xrightarrow{w_{i,j}} (\vec{\boldsymbol{q}}, [wz_{i,j}])$.

With the help of these properties we obtain an infinite run with the following labeling:

$$\gamma := u \left(\prod_{i \in P} \prod_{1 \leq j \leq n_i} v_{i,j} w_{i,j} \right)^{\omega},$$

which satisfies the accepting conditions of the table $\mathcal{T}$ in $\mathfrak{A}$, i.e., we have $\gamma \in L^{\omega}_{\mathcal{T}}(\mathfrak{A}) \neq \varnothing$.

Conversely, let $L^{\omega}_{\mathcal{T}}(\mathfrak{A}) \neq \varnothing$. Hence, there are $\alpha_0, \alpha_1, \alpha_2, \cdots \in \Gamma$, $\vec{\boldsymbol{q}}_{\boldsymbol{0}}, \vec{\boldsymbol{q}}_{\boldsymbol{1}}, \cdots \in \vec{\boldsymbol{Q}}$, and $x_0, x_1, \cdots \in A^*$ with $\alpha_0\alpha_1\alpha_2 \cdots \in L^{\omega}_{\mathcal{T}}(\mathfrak{A})$, $(\vec{\boldsymbol{q}}_{\boldsymbol{0}}, [x_0]) \in \mathrm{Init}_{\mathfrak{A}}$,

$$(\vec{\boldsymbol{q}}_{\boldsymbol{0}}, [x_0]) \xrightarrow{\alpha_0}_{\mathfrak{A}} (\vec{\boldsymbol{q}}_{\boldsymbol{1}}, [x_1]) \xrightarrow{\alpha_1}_{\mathfrak{A}} (\vec{\boldsymbol{q}}_{\boldsymbol{2}}, [x_2]) \xrightarrow{\alpha_2}_{\mathfrak{A}} \cdots,$$

and for each $i \in P$ and $1 \leq j \leq n_i$ there are infinitely many $h \in \mathbb{N}$ with $\vec{\boldsymbol{q}}_{\boldsymbol{h}}\restriction_i = f_{i,j}$. Since this is a run of $\mathfrak{A}$, there are transitions $(\vec{\boldsymbol{q}}_{\boldsymbol{h}}, \alpha_h, \overline{a_h} t_h, \vec{\boldsymbol{q}}_{\boldsymbol{h+1}}) \in \Delta$ with $[\![\overline{a_h} t_h]\!]([x_h]) = [x_{h+1}]$ for each $h \in \mathbb{N}$.

▷ *Idea.* To prove that the two conditions of our lemma hold, we first construct multiple new runs of $\mathfrak{A}$ as described in the proof of Lemma 9.5.30. Concretely, here we take all maximal letters $a \in A$ wrt. $\leq$ from the set of all letters occurring infinitely often in this run. For each of these maximal letters a we repeat the construction from above. Afterwards we merge these new runs together. This is possible since all of our new runs only modify the local states from $a\,\mathrm{M}$ after some steps. For this merged run, we can show the conditions as in the proof of Lemma 9.5.30. ⌟

Again, we define $A_k = \{a_h \mid h \geq k\} \subseteq A$ as the set of letters we read after the k^{th} step (for $k \in \mathbb{N}$). Then there is a number $k \in \mathbb{N}$, such that each letter $a \in A_k$ is read infinitely often on the infinite run from above. Due to the pigeon hole principle we can also choose k such that the subrun starting from the k^{th} step visits each state $\vec{\boldsymbol{q}}_{\boldsymbol{h}}$ infinitely often (for $h \geq k$). We already know that for each pair $a, b \in A_k$ of distinct letters, where a is maximal wrt. the well-quasi-ordering $\leq$, we have $b \leq a$ or $a \parallel b$. Hence, we can partition A into alphabets $B^{(0)}, B^{(1)}, \ldots, B^{(m)} \subseteq A$ such that $B^{(0)} := A \setminus A_k$ and for each $1 \leq n \leq m$ there is a maximal $a \in A_k$ such that $B^{(n)} := A_k \cap A_{\leq a}$. Note that this definition implies $a \parallel b$ for each $a \in B^{(n_1)}$ and $b \in B^{(n_2)}$ with $1 \leq n_1 < n_2 \leq m$.

Now, let $1 \leq n \leq m$ and $a \in B^{(n)}$ be maximal wrt. $\leq$. Similar to the proof of Lemma 9.5.30 we can construct a new run $\rho^{(n)} := (\vec{\boldsymbol{p}}^{\boldsymbol{(n)}}_{\boldsymbol{h}}, [y^{(n)}_h])_{h \in \mathbb{N}}$ in $\mathfrak{A}$ which starts in $(\vec{\boldsymbol{q}}_{\boldsymbol{k}}, [x_k])$, simulates $\mathfrak{A}$'s computations on the stacks in $a\,\mathrm{M}$ and leaves the stacks from $P \setminus a\,\mathrm{M}$ untouched. Then there is an infinite sequence $I^{(n)} := \left\{ h \geq k \;\middle|\; \forall \ell \geq h, i \in a\,\mathrm{M} \colon |y^{(n)}_h|_{A_i} \leq |y^{(n)}_\ell|_{A_i} \right\}$ of steps having minimal stack heights in $\rho^{(n)}$. We also find $\vec{\boldsymbol{p}}^{\boldsymbol{(n)}} \in \vec{\boldsymbol{Q}}$ and $b^{(n)} \in B^{(n)} = A_k \cap A_{\leq a} \subseteq A$ such that $J^{(n)} := \left\{ h \in I^{(n)} \;\middle|\; \vec{\boldsymbol{p}}^{\boldsymbol{(n)}} = \vec{\boldsymbol{p}}^{\boldsymbol{(n)}}_{\boldsymbol{h}}, [y^{(n)}_h] \in [b^{(n)}] \cdot \mathbb{M}(\mathcal{A}) \right\}$ is infinite. Let $h_1 \in J^{(n)}$ be arbitrary. Additionally, let $i \in a\,\mathrm{M}$ and $1 \leq j \leq n_i$. Since the path $(\vec{\boldsymbol{q}}_{\boldsymbol{h}}, [x_h])_{h \in \mathbb{N}}$ visits $f_{i,j}$ infinitely often and by construction of $\rho^{(n)}$, there are also infinitely many $h \in \mathbb{N}$ such that the i^{th} component of $\vec{\boldsymbol{p}}^{\boldsymbol{(n)}}_{\boldsymbol{h}}$ is $f_{i,j}$. Hence, there is $h_2 > h_1$ where $\vec{\boldsymbol{p}}^{\boldsymbol{(n)}}_{\boldsymbol{h_2}} \in (\vec{\boldsymbol{Q}}\restriction_{P \setminus \{i\}}, \{f_{i,j}\})$. Since

$J^{(n)}$ is infinite, there is $h_3 \in J^{(n)}$ with $h_3 > h_2$. Then we can prove (similar to the proof of Lemma 9.5.30)

$$\mathrm{pre}^*_{\mathfrak{A}}\big(\{\vec{\boldsymbol{p}}^{(n)}\} \times ([b^{(n)}] \cdot \mathbb{M}(\mathcal{A}))\big) \cap \mathrm{Init}_{\mathfrak{A}} \neq \varnothing \tag{9.6}$$

and

$$(\vec{\boldsymbol{p}}^{(n)}, [b^{(n)}]) \in \mathrm{pre}^+_{\mathfrak{A}}\big((\vec{\boldsymbol{Q}}\restriction_{P\setminus\{i\}}, \{f_{i,j}\}) \times \mathbb{M}(\mathcal{A}) \cap \mathrm{pre}^+_{\mathfrak{A}}(\{\vec{\boldsymbol{p}}^{(n)}\} \times ([b^{(n)}] \cdot \mathbb{M}(\mathcal{A})))\big). \tag{9.7}$$

Note that these equations also hold if we replace $\vec{\boldsymbol{p}}^{(n)}$ by any global state $\vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}$ with $\vec{\boldsymbol{q}}\restriction_{a\,\mathrm{M}} = \vec{\boldsymbol{p}}^{(n)}\restriction_{a\,\mathrm{M}}$ and $\vec{\boldsymbol{q}}\restriction_{P\setminus a\,\mathrm{M}} = \vec{\boldsymbol{q}}_{\boldsymbol{h}}\restriction_{P\setminus a\,\mathrm{M}}$ with $h \geq k$.

Finally, we set $\vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}$ such that $\vec{\boldsymbol{q}}\restriction_{B^{(0)}} = \vec{\boldsymbol{q}}_{\boldsymbol{k}}\restriction_{B^{(0)}}$ and $\vec{\boldsymbol{q}}\restriction_{B^{(n)}} = \vec{\boldsymbol{p}}^{(n)}\restriction_{B^{(n)}}$ for each $1 \leq n \leq m$. Additionally, we set $w := b^{(1)}b^{(2)}\dots b^{(m)}$. Then we obtain our claim by combining the equations (9.6) and (9.7) since $\mathfrak{A}$ is asynchronous and $B^{(n_1)}$ and $B^{(n_2)}$ are independent for each $1 \leq n_1 < n_2 \leq m$. ◂

Now, we want to take a closer look at the complexity of the emptiness problem: to solve this problem we can check for each state $\vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}$ and each word $w \in A^{\leq |P|}$ whether the properties of Lemma 9.5.31 hold. If we consider the underlying alphabet $\mathcal{A}$ as fixed, there are only a constant number of such words in $A^{\leq |P|}$. Hence, we have to check a constant number of properties linearly often. Any of these properties is decidable in polynomial time due to Theorem 9.5.9. This means that the emptiness problem is decidable in polynomial time in this case.

Next, we assume that the distributed alphabet $\mathcal{A}$ is part of the input of our considered emptiness problem. Then $A^{\leq |P|}$ contains exponentially many words of length polynomial in the size of $\mathcal{A}$. In this case our algorithm needs exponential time. However, due to the proof of Lemma 9.5.31 we do not have to check the properties of this lemma for all words from $A^{\leq |P|}$. It suffices to consider those words $w \in A^{\leq |P|}$ containing at most one letter from each process, i.e., those words with $|w|_{A_i} \leq 1$ for each $i \in P$. Though, in the worst case we still have to check $|2^P|$ many words implying an exponential running time.

Theorem 9.5.32. *The emptiness problem for ω-languages accepted by asynchronous Büchi-pushdown automata with global or local accepting conditions is decidable in polynomial time (if the underlying distributed alphabet is fixed).* ◂

Now, let ϕ be a TrPTL-formula and $\mathfrak{S}$ be an asynchronous pushdown system with local labeling. Then the constructed Büchi-automaton $\mathfrak{A}$ accepting all runs satisfying ϕ (cf. Chapter 8) is asynchronous. Therefore, the resulting Büchi-pushdown automaton $\mathfrak{B}$ simulating $\mathfrak{A}$ and $\mathfrak{S}$ in parallel also is asynchronous. If the underlying distributed alphabet $\mathcal{A}$ is fixed, we can check in polynomial time whether $\mathfrak{B}$ accepts at least one ω-word according to Theorem 9.5.32. Since the computation of $\mathfrak{A}$ takes exponential time, we can check $\mathcal{M}(\mathfrak{S}) \vDash \phi$ in exponential time. Hence, the model checking problem of TrPTL is decidable in EXPTIME.

Note that with TrPTL we can only state local properties of a run in an asynchronous pushdown system. One may also ask for global properties of such a run. It is well-known that global LTL for traces is decidable, but has non-elementary complexity [Wal98]. However, until now it is still an open problem, whether this result can be generalized to runs of asynchronous pushdown systems. Also the decidability of the model checking problem of the various temporal logics presented in [GK07] is unknown.

9.6 Conclusion

In this chapter we considered the reachability problem of pushdown automata having one or more stacks as their memory. We started with recalling the results from Bouajjani et al. [BEM97] and Finkel et al. [FWW97] proving that the reachability problem and the recurrent reachability problem is decidable in polynomial time in pushdown automata having one stack. Afterwards we considered automata having at least two stacks as their memory. Since these automata are known to be as powerful as Turing-machines, we considered a special kind of multi-pushdown automata, so-called *asynchronous pushdown automata*. We have translated the construction from [BEM97] from single-pushdown automata to the asynchronous ones and proved that in this case the reachability problem is still efficiently decidable. Additionally, we have presented a new construction considering the forwards reachable configurations. We have also seen that recurrent reachability with local or global restrictions is decidable in polynomial time (at least for a fixed underlying distributed alphabet). This finally implies the decidability of the model checking problem of a local LTL for traces in Kripke-structures induced by such asynchronous pushdown automata. So, we obtain that model checking of LTL-formulas for such Kripke-structures with local properties is in EXPTIME. For Kripke-structures with global properties it is still open whether the model checking problem is decidable.

Chapter 10

Verifying Queue Automata

10.1 Introduction

Our aim in this chapter is to verify automata having one or more queues as their memory. We can use such systems to model finite computer networks communicating through several channels. Unfortunately, an automaton having at least one reliable, unbounded queue is as powerful as a Turing-machine (cf. [BZ83]). This also holds for any partially lossy queue except for the fully lossy ones (cf. Theorem 3.3.7). This implies the undecidability of all non-trivial questions on verification like reachability or LTL model checking. In contrast, the reachability problem in lossy queue automata is decidable, but of very bad complexity. Concretely, given regular languages $K, L \subseteq A^*$ and $T \subseteq \Sigma^*$, we can decide whether we can reach from an initial queue content in L another queue content in K after application of T, i.e., whether $\textsc{Reach}(L, T) \cap K \neq \emptyset$ holds. Since $\textsc{Reach}(L, T)$ is downwards-closed under the subword relation, we know that this language is regular [Hai69]. However, in general we cannot compute an NFA accepting $\textsc{Reach}(L, T)$ from L and T [May03]. Surprisingly, from L and T we can compute an NFA accepting $\textsc{BackReach}(L, T)$ [AJ96a], even though this construction is not primitive recursive [Sch02, CS08]. Note that the recurrent reachability problem and the LTL model checking problem still are undecidable for lossy queue automata [AJ96b].

Due to the undecidability or inefficiency of the reachability problem in automata having at least one partially lossy queue, it is worth to approximate this problem. A first, very trivial approach is to simulate the computation of a queue automaton step-by-step. This means, starting with L, we iteratively compute the set of all queue contents which are reachable from L after application of at most n transitions. Formally, for the set $T_n \subseteq \Sigma^*$ of prefixes of length at most n of T we compute $L_n = \textsc{Reach}(L, T_n)$ for increasing numbers $n \in \mathbb{N}$. This can be done by applying single basic actions $\alpha \in \Sigma$ to L_{n-1} which corresponds to concatenation or taking a left-quotient. So, if L is accepted by a finite automaton (in literature these are often called *queue decision diagrams* or *QDDs*, for short, cf. [Boi+97, BG99, KM21]), all languages $L_n \subseteq A^*$ are regular and automata accepting these languages can be computed in polynomial time using the classical constructions from automata theory. However, this algorithm is not very efficient: consider $L = \{\varepsilon\}$ and $T = a^*$ for an $a \in A$. Then we obtain $L_n = \textsc{Reach}(L, a^{\leq n}) = a^{\leq n}$ which is still finite for each $n \in \mathbb{N}$. In other words, after a finite number of steps we can only explore a finite extension of the state space L.

Boigelot et al. [Boi+97, BG99] and Abdulla et al. [Abd+04] improved this trivial approximation by the introduction of so-called *meta-transformations*. These are certain regular languages $S \subseteq \Sigma^*$ such that the sets $\textsc{Reach}(L, S)$ are effectively regular for any regular language $L \subseteq A^*$. Then the trivial approximation can be modified as follows: whenever we

compute L_{n+1} from L_n we search for such meta-transformation in the queue automaton's control component starting from T_n and apply them to L_n. In [Boi+97] the authors considered meta-transformations of the form $S = \{t\}$ and $S = \{t\}^*$ where $t \in \Sigma^*$. In fact, this approach is more efficient than the trivial one, since we can explore an infinite state space in just one step of the algorithm.

From these results we obtain the decidability of the reachability problem in so-called *flat* (or *flattable*) queue automata. These are plq automata where each control state belongs to at most one loop. In this case, the algorithm from Boigelot et al. terminates after a finite number of steps and $\textsc{Reach}(L, T)$ is regular whenever L and T are regular languages. Moreover, due to [FP20, Sch20] the membership problem of $\textsc{Reach}(L, T)$ is NP-complete in flat reliable and lossy queue automata. Bollig et al. proposed to extend this class to so-called *input-boundable* queue automata [BFS20]. A regular language $T \subseteq \Sigma^*$ of queue action sequences is *input-bounded* if $\mathrm{wrt}(T) \subseteq w_1^* w_2^* \dots w_n^*$ and $\mathrm{rd}(T)$ contains only prefixes of $w_1^* w_2^* \dots w_n^*$ where $w_1, \dots, w_n \in A^+$ are non-empty words. In that paper, the authors reduced the reachability problem to reachability in n-counter automata. The result is that membership of $\textsc{Reach}(\{\varepsilon\}, T)$ is decidable, but it is not known whether there is a primitive recursive algorithm solving this problem (recall that the reachability problem of n-counter automata has non-primitive recursive complexity [CO22, Ler22]). Note that this decidability is still true for automata having more than one queue.

In the next section we present some more meta-transformations in reliable queue automata having exactly one queue. Afterwards, we translate these results to arbitrary partially lossy queue automata and to distributed queue automata.

10.2 Reliable Queues[xviii]

In this section we want to consider the reachability problem of automata having only one reliable and unbounded queue as their memory. First, we show that the mappings $\textsc{Reach}$ and $\textsc{BackReach}$ are dual. Due to this duality it will suffice later in this section to only consider forwards reachability. Using this duality will finally yield similar results for the backwards reachability problem. Before we state this duality, we first should recall the duality function $\mathrm{d}\colon \Sigma^* \to \Sigma^*$. This function is defined as follows:

$$\mathrm{d}(\varepsilon) = \varepsilon, \quad \mathrm{d}(at) = \mathrm{d}(t)\overline{a}, \text{ and } \quad \mathrm{d}(\overline{a}t) = \mathrm{d}(t)a$$

for each $a \in A$ and $t \in \Sigma^*$. We already know that $[\![t]\!](v) = w$ holds if, and only if, $[\![\mathrm{d}(t)]\!](w^{\mathrm{R}}) = v^{\mathrm{R}}$ holds for each $v, w \in A^*$ and $t \in \Sigma^*$ according to Proposition 4.6.1. This equivalence turns out to be very helpful when proving the duality of the forwards and backwards reachability in queue systems:

Theorem 10.2.1. *Let A be an alphabet, $L \subseteq A^*$, and $T \subseteq \Sigma^*$. Then we have*

$$\textsc{BackReach}(L, T) = \textsc{Reach}(L^{\mathrm{R}}, \mathrm{d}(T))^{\mathrm{R}}.$$

[xviii] The results of this section are published in [Köc19, Köc21].

Proof. First, let $w \in \textsc{BackReach}(L, T)$. By definition there are $t \in T$ and $v \in L$ with $v = [\![t]\!](w)$. By Proposition 4.6.1(1) we infer that $[\![\mathrm{d}(t)]\!](v^{\mathrm{R}}) = w^{\mathrm{R}}$ holds. This implies $w^{\mathrm{R}} \in \textsc{Reach}(L^{\mathrm{R}}, \mathrm{d}(T))$ and, hence, $w \in \textsc{Reach}(L^{\mathrm{R}}, \mathrm{d}(T))^{\mathrm{R}}$.

Conversely, let $w \in \textsc{Reach}(L^{\mathrm{R}}, \mathrm{d}(T))^{\mathrm{R}}$, i.e., $w^{\mathrm{R}} \in \textsc{Reach}(L^{\mathrm{R}}, \mathrm{d}(T))$. Then there are $v \in L$ and $t \in T$ with $[\![\mathrm{d}(t)]\!](v^{\mathrm{R}}) = w^{\mathrm{R}}$. Due to Proposition 4.6.1(1) we have $[\![t]\!](w) = v \in L$. By definition we have $w \in \textsc{BackReach}(L, T)$. ◂

In the following we always consider sets of queue contents which are forwards or backwards reachable from a given regular language of queue contents via a regular language of action sequences. Since the regular languages are closed under reversal and duality d and since forwards and backwards reachability are dual, it suffices to consider the forwards reachable queue contents, only.

Now, let $L \subseteq A^*$ be a recursively enumerable language of queue contents and $T \subseteq \Sigma^*$ be a regular language of queue action sequences. Then the language $\textsc{Reach}(L, T)$ is effectively recursively enumerable. However, since queue systems can simulate Turing-machines, the language $\textsc{Reach}(L, T)$ can be any recursively enumerable language - even if L and T are somewhat "simple" languages:

Remark 10.2.2. Let $K \subseteq \Gamma^*$ be a recursively enumerable language. Then there is a (type-0) grammar $\mathfrak{G} = (N, \Gamma, P, S)$ with $K = L(\mathfrak{G})$. Let $\# \notin N \cup \Gamma$ be a new letter. We set our alphabet $A := N \cup \Gamma \cup \{\#\}$ (this will be the set of possible entries in our queue). We construct the set of action sequences $T \subseteq \Sigma^*$ as follows:

$$T := \left(\{\overline{\ell} r \mid (\ell, r) \in P\} \cup \{\overline{a} a \mid a \in A\}\right)^* \cdot \overline{\#},$$

i.e., the queue system can apply any rule from $\mathfrak{G}$ and move any letter from the head to its tail. Then we have

$$\textsc{Reach}(\{S\#\}, T) \cap \Gamma^* = L(\mathfrak{G}) = K. \qquad ⌟$$

In other words, $\textsc{Reach}(L, T)$ can be any recursively enumerable language K even if L is a singleton and T is essentially the Kleene-closure of a finite set of action sequences. Hence, $\textsc{Reach}(L, T)$ can be undecidable even for such simple languages L and T. Therefore, we need the approximation of $\textsc{Reach}(L, T)$ using meta-transformations as mentioned above.

In the following proposition we consider a first class of simple meta-transformations. Concretely, we consider the case that T contains only sequences of write actions or only sequences of read actions:

Proposition 10.2.3. *Let A be an alphabet and $L, T \subseteq A^*$. Then the following statements hold:*

(1) $\textsc{Reach}(L, T) = LT$ *and*

(2) $\textsc{Reach}(L, \overline{T}) = {}_T\backslash L$.

Note that this proposition is a generalization of Theorems 1 and 2 in [BG99]. Concretely, in these two theorems Boigelot and Godefroid have proven the effective regularity of $\textsc{Reach}(L, t)$ and $\textsc{Reach}(L, \overline{t})$ where $L \subseteq A^*$ is regular and $t \in A^*$ is a single action sequence.

Proof. First, consider equation (1): let $w \in \textsc{Reach}(L, T) = [\![T]\!](L) \setminus \{\bot\}$. Then there are $v \in L$ and $t \in T$ with $[\![t]\!](v) = w$. Since $t \in T \subseteq A^*$ we have, by Observation 3.2.2, $w = [\![t]\!](v) = vt \in LT$. Conversely, let $w \in LT$. Then there is $v \in L$ and $t \in T$ with $vt = w$. By iterated application of the definition of $[\![a]\!]$ (for $a \in A$) we obtain $w = vt = [\![t]\!](v) \in [\![T]\!](L)$. Since $L, T \subseteq A^*$ and $w \neq \bot$ we obtain $w \in [\![T]\!](L) \setminus \{\bot\} = \textsc{Reach}(L, T)$.

Similarly, we can prove equation (2) using the definition of $[\![\overline{a}]\!]$ for $a \in A$. ◂

Combining Theorem 10.2.1 and Proposition 10.2.3 we obtain the following two equations:

$$\textsc{BackReach}(L, T) = \textsc{Reach}(L^{\mathrm{R}}, \overline{T}^{\mathrm{R}})^{\mathrm{R}} = \left({}_{T^{\mathrm{R}}}\backslash L^{\mathrm{R}}\right)^{\mathrm{R}} = L/_T \qquad \text{and}$$
$$\textsc{BackReach}(L, \overline{T}) = \textsc{Reach}(L^{\mathrm{R}}, T^{\mathrm{R}})^{\mathrm{R}} = \left(L^{\mathrm{R}} T^{\mathrm{R}}\right)^{\mathrm{R}} = TL$$

for each pair of languages $L, T \subseteq A^*$.

Now, let $L, T \subseteq A^*$ be two regular languages accepted by the NFAs $\mathfrak{L}$ and $\mathfrak{T}$, respectively. Then, using the classical constructions, we can construct an NFA accepting $\textsc{Reach}(L, T) = LT$ in quadratic time. An NFA accepting $\textsc{Reach}(L, \overline{T}) = {}_T\backslash L$ can be constructed in cubic time. The number of states of these two NFAs is linear in the number of states in $\mathfrak{L}$ and $\mathfrak{T}$.

If we require these languages to be accepted by DFAs, then we additionally need to determinize the given NFAs resulting in exponential time and size. The complexities regarding BackReach are similar to the ones regarding Reach. Though, if $\mathfrak{L}$ is a DFA, we still can compute a DFA accepting $\textsc{BackReach}(L, T) = L/_T$ in cubic time having a linear number of states (in this case we only modify the accepting states of $\mathfrak{L}$).

Next, we consider some more complicated types of meta-transformations T having mappings $L \mapsto \textsc{Reach}(L, T)$ and $L \mapsto \textsc{BackReach}(L, T)$ which preserve regularity efficiently.

10.2.1 Recognizability

The first type of meta-transformations we want to consider are languages that are regular and closed under the behavioral equivalence $\equiv$. In other words, we consider pre-images of recognizable languages in the queue monoid.

Let $t \in \Sigma^*$ be an action sequence. From Corollary 4.7.15 we know that there is a "simple" sequence $s \in \overline{A}^* A^* \overline{A}^*$ with $s \equiv t$. In particular, we have $t \equiv \overline{\mathrm{rd}_1(t)}\,\mathrm{wrt}(t)\,\overline{\mathrm{rd}_2(t)}$. Hence, we can compute this "simple" action sequence in polynomial time. To this end, we have to compute the normal form $\mathrm{nf}(t)$ of t. From this normal form we finally can extract the words $\mathrm{rd}_1(t)$, $\mathrm{wrt}(t)$, and $\mathrm{rd}_2(t)$. Note that, even if this algorithm computes a unique simple word from its input t, there are action sequences having multiple such simple, equivalently behaving action sequences $s \in \overline{A}^* A^* \overline{A}^*$ as the following example shows:

Example 10.2.4. Let $a, b \in A$ be two distinct letters and $t = a\overline{a}\overline{b}a$. Then from Corollary 4.7.15 we obtain $\overline{a}\overline{b}aa \equiv t$, but we also have $\overline{a}aa\overline{b} \equiv aa\overline{a}\overline{b} \equiv t$. ⌟

Now, let $L \subseteq A^*$ and $T \subseteq \Sigma^*$ be two regular languages such that T is closed under behavioral equivalence. In other words, we have $T = \eta^{-1}(S)$ for a recognizable queue language (for characterizations of these languages see Section 6.4). As we have mentioned, for each $t \in T$ there is another action sequence $s \in T$ which is "simple" (i.e., $s \in \overline{A}^* A^* \overline{A}^*$) and behaves equivalently to t. Since T is closed under $\equiv$ we also have $s \in T$ in this case. Hence, to compute $\textsc{Reach}(L, T)$ it suffices to consider the simple action sequences, only. Therefore, we obtain the following result:

Theorem 10.2.5. *Let A be an alphabet, $L \subseteq A^*$, and $T \subseteq \Sigma^*$ be regular languages where T is closed under $\equiv$ (i.e., we have $T = \eta^{-1}(S)$ for a recognizable queue language $S \subseteq \mathbb{T}(Q_A)$). Then $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$ are regular. In particular, we can construct NFAs accepting $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$, respectively, from NFAs accepting L and T in polynomial time.*

Proof. First, we prove the following equation:

▷ *Claim.* *We have* $\textsc{Reach}(L, T) = \textsc{Reach}(L, T \cap \overline{A}^* A^* \overline{A}^*)$.

Proof. The inclusion "$\supseteq$" is trivial since $T \cap \overline{A}^* A^* \overline{A}^* \subseteq T$ and $\textsc{Reach}(L, .)$ is monotonic. Towards the converse inclusion, let $w \in \textsc{Reach}(L, T)$. Then there are $v \in L$ and $t \in T$ with $[\![t]\!](v) = w$. Due to Corollary 4.7.15 there is $s \in \overline{A}^* A^* \overline{A}^*$ with $s \equiv t$, i.e., we have $s \in T$. Since $s \in T \cap \overline{A}^* A^* \overline{A}^*$ we obtain $w = [\![t]\!](v) = [\![s]\!](v) \in \textsc{Reach}(L, T \cap \overline{A}^* A^* \overline{A}^*)$ by the definition of $\equiv$. ◁

Next, we compute $\textsc{Reach}(L, T \cap \overline{A}^* A^* \overline{A}^*)$. To this end, let $\mathfrak{T} = (Q, \Sigma, I, \Delta, F)$ be an NFA with $L(\mathfrak{T}) = T$. We partition $T \cap \overline{A}^* A^* \overline{A}^*$ as follows: let $p, q \in Q$ be any pair of states of $\mathfrak{T}$. Then we can compute the following three regular languages in polynomial time:

(1) $\overline{K_1^{p,q}} := L(\mathfrak{T}_{I \to p}) \cap \overline{A}^*$,

(2) $K_2^{p,q} := L(\mathfrak{T}_{p \to q}) \cap A^*$, and

(3) $\overline{K_3^{p,q}} := L(\mathfrak{T}_{q \to F}) \cap \overline{A}^*$.

It is easy to see that $T \cap \overline{A}^* A^* \overline{A}^* = \bigcup_{p,q \in Q} \overline{K_1^{p,q}} K_2^{p,q} \overline{K_3^{p,q}}$ holds. Hence, we have

$$\textsc{Reach}(L, T) = \textsc{Reach}\left(L, \bigcup_{p,q \in Q} \overline{K_1^{p,q}} K_2^{p,q} \overline{K_3^{p,q}}\right) = \bigcup_{p,q \in Q} \textsc{Reach}\left(L, \overline{K_1^{p,q}} K_2^{p,q} \overline{K_3^{p,q}}\right).$$

So, let $p, q \in Q$. By Proposition 10.2.3 reading from the queue corresponds to taking the left-quotient and writing into the queue corresponds to concatenation. Therefore, we have:

$$\textsc{Reach}\left(L, \overline{K_1^{p,q}} K_2^{p,q} \overline{K_3^{p,q}}\right) = {}_{K_3^{p,q}}\backslash\left(\left({}_{K_1^{p,q}}\backslash L\right) \cdot K_2^{p,q}\right).$$

Hence, due to the closure properties of the class of regular languages, $\textsc{Reach}(L, T)$ is effectively regular. Since all of the needed closure properties are also efficient and since we are considering only $O(|Q|^2)$ many languages $K_i^{p,q}$, an NFA accepting $\textsc{Reach}(L, T)$ can be computed in polynomial time. This NFA has a cubic number of states.

Finally, we have to show that $\textsc{BackReach}(L, T)$ is efficiently regular. Recall that, by Theorem 10.2.1, we have $\textsc{BackReach}(L, T) = \textsc{Reach}(L^R, \mathrm{d}(T))^R$. Due to Proposition 4.6.1(2) the language $\mathrm{d}(T)$ is still closed under behavioral equivalence. Additionally, $\mathrm{d}(T)$ is effectively regular since we only have to replace a by $\overline{a}$ and vice versa and to invert the edges of the NFA accepting T. Since the class of regular languages is efficiently closed under reversal, we can compute an NFA accepting $\textsc{BackReach}(L, T)$ in polynomial time which has a cubic number of states. ◂

Note that for a given regular language $T \subseteq \Sigma^*$ it is decidable, whether T is closed under behavioral equivalence (cf. Theorem 5.4.8). However, there are regular languages $T \subseteq \Sigma^*$ such that the closure under behavioral equivalence is not regular. According to Theorem 5.4.7(2) it is not even decidable whether the closure under behavioral equivalence of a given regular language $T \subseteq \Sigma^*$ is regular again.

Example 10.2.6. Let $a \in A$. Then the closure T of $(a\overline{a})^*$ under behavioral equivalence is not regular since

$$T \cap \overline{A}^* A^* \overline{A}^* = \{a^n \overline{a}^n \mid n \in \mathbb{N}\}$$

is no regular language. ⌟

Due to the proof of Theorem 10.2.5 the map $L \mapsto \textsc{Reach}(L, T)$ preserves regularity also for each regular language $T \subseteq \overline{A}^* A^* \overline{A}^*$. It is also possible to extend the result to context-free languages of queue contents:

Theorem 10.2.7. *Let A be an alphabet, $L \subseteq A^*$ be context-free, and $T \subseteq \Sigma^*$ be regular and closed under $\equiv$. Then $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$ are context-free. In particular, we can construct PDAs accepting $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$, respectively, from a PDA accepting L and an NFA accepting T in polynomial time.*

Proof idea. This is similar to the proof of Theorem 10.2.5 due to the efficient closure properties of context-free languages (recall that the left or right quotient of a context-free language wrt. a regular language is context-free, again). ◂

10.2.2 Alternations

Now, we want to consider another kind of meta-transformations: cyclic regular languages. In other words, given two regular languages $L \subseteq A^*$ and $T \subseteq \Sigma^*$, we want to compute an NFA accepting $\textsc{Reach}(L, T^*)$. The case, where T is a singleton, was first considered by Boigelot et al. in [Boi+97, BG99] (and similarly by Abdulla et al. in [Abd+04] for lossy queues). In those papers the authors proved that $\textsc{Reach}(L, T^*)$ is effectively regular in this case. So, a natural question would be to ask, whether this result also holds if T contains more than only one action sequence. In practice, this is relevant for transactional systems consisting of a single database server and multiple clients. Then the set $T \subseteq \Sigma^*$ consists of finite sequences of tasks a client sends to the server (this corresponds to write actions into the server's task queue) and finite sequences of tasks the server executes (these are read actions from its task queue).

Unfortunately, we have seen in Remark 10.2.2 that $\textsc{Reach}(L, T^*)$ can be undecidable even if T is a finite language. The following example proves that $\textsc{Reach}(L, T^*)$ is not necessarily regular anymore even if T consists of only two action sequences:

Example 10.2.8. Let A be an alphabet and $a, b \in A$ be two distinct letters. Then we have

$$\textsc{Reach}(\{a\}, \{\overline{a}bb, \overline{b}a\}^*) \cap a^* = \{a^{2^n} \mid n \in \mathbb{N}\}$$

which is not even context-free. ⌟

In both cases, Remark 10.2.2 and Example 10.2.8, the write actions of any sequence $t \in T$ depend on the read actions in t. So, we are able to copy data from the head of the queue into its tail. Then we can see the queue as a Turing-tape and we are able to move the head on this tape in any direction. Hence, to find languages $T \subseteq \Sigma^*$ such that $L \mapsto \textsc{Reach}(L, T^*)$ preserves regularity, we should consider those regular languages $T \subseteq \Sigma^*$ of action sequences in which the write actions do not depend on the read actions. This means, we have for any pair $s, t \in T$ another action sequence $r \in T$ consisting of the write actions from s and the read actions from t. Then, independently of the word from $\mathrm{wrt}(T)$ we write into the queue, we can read any word from $\mathrm{rd}(T)$. Formally, we are considering the following sets of action sequences:

Definition 10.2.9. Let A be an alphabet. A language $T \subseteq \Sigma^*$ is *read-write independent* if, for each $s, t \in T$, we have $\mathrm{wrt}(s)\overline{\mathrm{rd}(t)} \in T$. In other words, T is read-write independent if, and only if, $\mathrm{wrt}(T)\overline{\mathrm{rd}(T)} \subseteq T$ holds. ⌟

We may see a read-write independent language T as some kind of a Cartesian product of a set of sequences of write actions $W \subseteq A^*$ with a set of read action sequences $\overline{R} \subseteq \overline{A}^*$ where for each pair $(w, \bar{r}) \in W \times \overline{R}$ we have the action sequence $w\bar{r} \in T$. Some simple read-write independent languages are given in the following example:

Example 10.2.10. Let $W, R \subseteq A^*$. Then $W\overline{R}$ and $W \sqcup\!\sqcup \overline{R}$ are read-write independent. ⌟

Since the class of regular languages is closed under projections and concatenation and due to the decidability of the inclusion problem, we can decide whether a given regular language $T \subseteq \Sigma^*$ is read-write independent.

For our further considerations of read-write independent languages we need the following lemma. It states that we can "de-shuffle" those languages:

Lemma 10.2.11. *Let A be an alphabet, $L \subseteq A^*$, and $T \subseteq \Sigma^*$ be read-write independent. Then we have*

$$\textsc{Reach}(L, T) = \textsc{Reach}(L, \mathrm{wrt}(T)\overline{\mathrm{rd}(T)}) .$$

Proof. The inclusion "$\supseteq$" is obvious since $\mathrm{wrt}(T)\overline{\mathrm{rd}(T)} \subseteq T$ and $\textsc{Reach}(L, .)$ is monotonic. Towards the converse inclusion, let $w \in \textsc{Reach}(L, T) = [\![T]\!](L) \setminus \{\bot\}$. Then there are $t \in T$ and $v \in L$ with $[\![t]\!](v) = w \neq \bot$. By Lemma 4.7.3 we have $\bot \neq w = [\![t]\!](v) = [\![\mathrm{wrt}(t)\overline{\mathrm{rd}(t)}]\!](v)$ implying $w \in \textsc{Reach}(L, \mathrm{wrt}(T)\overline{\mathrm{rd}(T)})$. ◂

Note that Lemma 10.2.11 does not hold for arbitrary languages $T \subseteq \Sigma^*$. For example, consider $L = \{\varepsilon\}$ and $T = \{\overline{a}a\}$. Then we know $[\![\overline{a}a]\!](\varepsilon) = \bot$ and $[\![a\overline{a}]\!](\varepsilon) = \varepsilon$ resulting in $\textsc{Reach}(\{\varepsilon\}, \{\overline{a}a\}) = \varnothing \subsetneq \{\varepsilon\} = \textsc{Reach}(\{\varepsilon\}, \{a\overline{a}\})$. However, due to Lemma 4.7.3 the following inequation holds for any language $T \subseteq \Sigma^*$ - even if T is not read-write independent:

$$\textsc{Reach}(L, T) \subseteq \textsc{Reach}(L, \mathrm{wrt}(T)\overline{\mathrm{rd}(T)}) .$$

Now, consider the language $T := \mathrm{wrt}(t) \sqcup\!\sqcup \overline{\mathrm{rd}(t)}$ for an action sequence $t \in \Sigma^*$. This language is read-write independent. Due to Theorem 4.7.8 T is also closed under behavioral equivalence, i.e., we can compute $\textsc{Reach}(L, T)$ in polynomial time by Theorem 10.2.5.

However, T^* is not necessarily closed under $\equiv$. Hence, we cannot apply Theorem 10.2.5 to compute $\textsc{Reach}(L, T^*)$. But from Lemma 10.2.11, $\mathrm{wrt}(T) = \mathrm{wrt}(t)$, and $\mathrm{rd}(T) = \mathrm{rd}(t)$ we learn

$$\textsc{Reach}(L, T^*) = \textsc{Reach}(L, (\mathrm{wrt}(T)\overline{\mathrm{rd}(T)})^*) = \textsc{Reach}(L, (\mathrm{wrt}(t)\overline{\mathrm{rd}(t)})^*) .$$

Since $\mathrm{wrt}(t)\overline{\mathrm{rd}(t)} \in \Sigma^*$ holds, the map $L \mapsto \textsc{Reach}(L, T^*)$ preserves regularity effectively by [Boi+97].

In the following, we will prove that, provided T is any regular and read-write independent language, the mapping $L \mapsto \textsc{Reach}(L, T^*)$ preserves regularity effectively and even efficiently (cf. Theorem 10.2.18). By Lemma 10.2.11 it suffices to consider languages of the form $T = W\overline{R}$ where $W, R \subseteq A^*$ are two regular languages. But before we show this general case, we make some additional assumptions on these languages W and R. Afterwards we derive the general case from this particular case. Concretely, we consider regular languages $W\overline{R} \subseteq A^*\overline{A}^*$ where A is an alphabet having a distinguished letter \$ which marks the beginning of each word from W and is used for synchronization between write and read actions. In this connection, we have to ensure that the \$ can be read whenever it occurs on the queue's head position. We do this by insertion of arbitrarily many $\overline{\$}'s$ at any position in T. In other words, we require $R = \$^* \sqcup\!\sqcup R$.

Theorem 10.2.12. *Let A be an alphabet and $\$ \in A$ be a letter. Additionally, let $L \subseteq (A \setminus \{\$\})^*$, $W \subseteq \$(A \setminus \{\$\})^*$, and $R \subseteq A^*$ be regular languages such that $R = \$^* \sqcup\!\sqcup R$ holds. Then $\textsc{Reach}(L, (W\overline{R})^*)$ is regular. In particular, we can construct an NFA accepting $\textsc{Reach}(L, (W\overline{R})^*)$ from NFAs accepting L, W, and R in polynomial time.*

The correctness proof of this theorem is based on two ideas:

(1) We delay decisions: instead of deciding which word $w \in W$ we write into the queue in each round, we just keep one token per round meaning that we have written a word. This token is represented by the letter \$ at the first position of the word w. When reading a word $r \in R$, we remove such tokens and decide in this moment, which word we have written into the queue before.

(2) Since we keep only tokens (of which we have only one type \$), our queue alphabet consists of only one letter. As we already know, such unary queue is equivalent to a counter or a unary pushdown. For technical reasons we see this storage mechanism as a unary pushdown.

Hence, our queue system applying $(W\overline{R})^*$ to its queue contents L can be simulated by a pushdown automaton $\mathfrak{P}$. For such automata the map $\mathrm{post}^*_{\mathfrak{P}} \colon \mathrm{Conf}_{\mathfrak{P}} \to \mathrm{Conf}_{\mathfrak{P}}$ effectively preserves regularity (in polynomial time) according to Theorem 9.2.1. This will finally yield the efficient regularity of $\textsc{Reach}(L, (W\overline{R})^*)$.

In the following we first describe the construction of the pushdown automaton. Afterwards, on Page 183, we prove the correctness of this theorem.

The Reduction to Pushdown Automata

Towards the simulation of our queue system with the help of a (unary) pushdown automaton, we first need an abstract presentation of the queue system's configurations. To this end, we consider a non-failing computation $t \in (W\overline{R})^*$ of the queue system with initial content $v \in L$, i.e., $[\![t]\!](v) \neq \bot$. So, there are intermediate queue contents $v_0, \ldots, v_m \in A^*$ and basic actions $\alpha_1, \ldots, \alpha_m \in \Sigma$ with $v_0 = v$, $v_{i+1} = [\![\alpha_{i+1}]\!](v_i) \neq \bot$ for each $0 \leq i < m$, and $\alpha_1 \ldots \alpha_m = t \in (W\overline{R})^*$. By $v_{i+1} = [\![\alpha_{i+1}]\!](v_i)$ we obtain $v_i \operatorname{wrt}(\alpha_{i+1}) = \operatorname{rd}(\alpha_{i+1})\, v_{i+1}$ for each $0 \leq i < m$ (by Observation 3.2.2). Hence, we have

$$v_0 \operatorname{wrt}(t) = \operatorname{rd}(t)\, v_m \,.$$

Since $t \in (W\overline{R})^*$ holds, we have $\operatorname{wrt}(t) \in W^*$ and, therefore, $v_0 \operatorname{wrt}(t) \in LW^*$. Let $\mathfrak{C}$[xix] be an NFA accepting the regular language LW^*. Then there is an accepting run $p_0, \ldots, p_\ell$ in $\mathfrak{C}$ labeled with $v_0 \operatorname{wrt}(t)$.

Due to closure properties, the language $(W\overline{R})^*$ is regular. Let $\mathfrak{T}$[xx] be an NFA accepting this language. Then there is an accepting run $s_0, \ldots, s_m$ in $\mathfrak{T}$ labeled with $t = \alpha_1 \ldots \alpha_m$.

Now, we want to abstract the configurations (s_i, v_i) of our queue system with the help of the following information:

(1) the state s_i from $\mathfrak{T}$ which is the control state of our queue system,

(2) two states p_{x_i} and p_{y_i} from $\mathfrak{C}$ such that $p_{x_i}, \ldots, p_{y_i}$ is a run of $\mathfrak{C}$ labeled with v_i, and

(3) the natural number $|v_i|_\$$ representing the number of words from W to be contained in v_i.

Initially, we abstract (s_0, v_0) by $(s_0, p_0, p_{|v|}, 0)$ since $p_0, \ldots, p_{|v|}$ is a run in $\mathfrak{C}$ labeled with $v_0 = v$ and $|v_0|_\$ = |v|_\$ = 0$ by $v \in L \subseteq (A \setminus \{\$\})^*$. Next, we can obtain the abstraction of (s_{i+1}, v_{i+1}) from (s_i, v_i) as follows: let $(s_i, p_{x_i}, p_{y_i}, n_i)$ be the abstraction of (s_i, v_i). By the choice of our run in $\mathfrak{T}$ we have an edge $s_i \xrightarrow{\alpha_{i+1}}_{\mathfrak{T}} s_{i+1}$. Additionally, we have to distinguish the following two cases:

(1) If $\alpha_{i+1} = a \in A$ is a write action, there is an edge $p_{y_i} \xrightarrow{a}_{\mathfrak{C}} p_{y_i+1}$. Since the run $p_{x_i}, \ldots, p_{y_i}$ is labeled by v_i, we can extend it to an $v_i a = v_{i+1}$-labeled run $p_{x_i}, \ldots, p_{y_i}, p_{y_i+1}$. Additionally, if $a = \$$ then the number of $\$$'s in v_i will be increased. Hence, we abstract (s_{i+1}, v_{i+1}) by $(s_{i+1}, p_{x_i}, p_{y_i+1}, n_i + |a|_\$)$.

(2) If $\alpha_{i+1} = \overline{a} \in \overline{A}$ is a read action, we have $v_i = a v_{i+1}$ implying that the run $p_{x_i}, \ldots, p_{y_i}$ labeled with v_i starts with the edge $p_{x_i} \xrightarrow{a}_{\mathfrak{C}} p_{x_i+1}$. Hence, $p_{x_i+1}, \ldots, p_{y_i}$ is a v_{i+1}-labeled run of $\mathfrak{C}$. If $a = \$$ then the number of $\$$'s in v_i decreases. The resulting abstraction of (s_{i+1}, v_{i+1}) in this case is $(s_{i+1}, p_{x_i+1}, p_{y_i}, n_i - |a|_\$)$.

All in all, $(s_i, p_{x_i}, p_{y_i}, n_i)$ is an abstraction of the queue system's configuration (s_i, v_i). These information can be simulated with the help of a pushdown automaton $\mathfrak{P}$. In this case, the control states of $\mathfrak{P}$ are composed of the states s_i, p_{x_i}, and p_{y_i} and the stack contains $\$^{n_i}$. Note that this PDA is essentially a (partially blind) one-counter automaton, but due to technical reasons we will utilize this more powerful automata model.

[xix] The letter $\mathfrak{C}$ indicates that this NFA accepts all possible queue Contents.

[xx] The letter $\mathfrak{T}$ indicates that this NFA accepts all possible Transformations our queue system can apply.

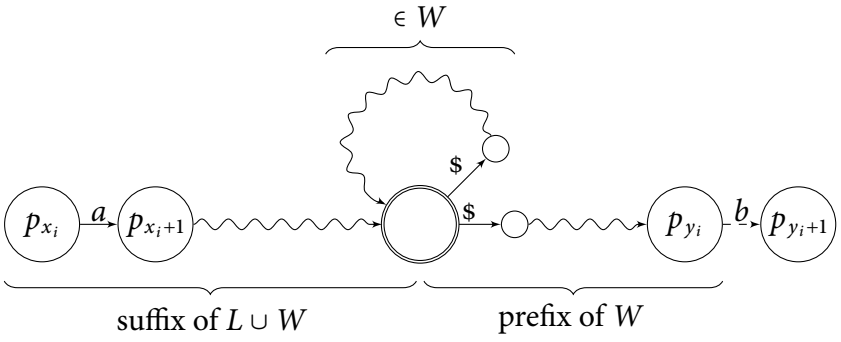

Figure 10.1. A run labeled with v_i from p_{x_i} to p_{y_i} in $\mathfrak{C}$.

Now, let $\mathfrak{C} = (Q_\mathfrak{C}, A, I_\mathfrak{C}, \Delta_\mathfrak{C}, F_\mathfrak{C})$ be an NFA accepting LW^* and $\mathfrak{T} = (Q_\mathfrak{T}, \Sigma, I_\mathfrak{T}, \Delta_\mathfrak{T}, F_\mathfrak{T})$ be an NFA accepting $(W\overline{R})^*$. W.l.o.g., we can assume that both NFAs, $\mathfrak{C}$ and $\mathfrak{T}$, are *trim* in the sense that each state is reachable from their initial states and can reach a final state. Additionally, we assume that $\mathfrak{C}$ and $\mathfrak{T}$ have exactly one final state called $f_\mathfrak{C}$ and $f_\mathfrak{T}$, respectively. Note that we can compute these two automata in polynomial time from NFAs accepting L, W, and R.

Recall that the queue system's configurations are abstracted by states from $\mathfrak{C}$ and $\mathfrak{T}$ and by a natural number. This can be simulated by the PDA $\mathfrak{P} = (Q_\mathfrak{P}, \Sigma, \mathcal{P}_{\{\$\}}, I_\mathfrak{P}, \varepsilon, \Delta_\mathfrak{P}, F_\mathfrak{P})$ which is defined as follows:

- $Q_\mathfrak{P} := Q_\mathfrak{T} \times Q_\mathfrak{C} \times Q_\mathfrak{C}$. Here, the first component represents the control state of the queue system. The second and third component represent two states characterizing the queue's content as described above.
- $I_\mathfrak{P} := I_\mathfrak{T} \times I_\mathfrak{C} \times Q_L$ where $Q_L := \{q \in Q_\mathfrak{C} \mid \exists v \in L\colon I_\mathfrak{C} \xrightarrow{v}_\mathfrak{C} q\}$.
- $F_\mathfrak{P} := F_\mathfrak{T} \times Q_\mathfrak{C} \times F_\mathfrak{C}$.
- $\Delta_\mathfrak{P}$ consists of the following transitions for $a \in A$, $s, s' \in Q_\mathfrak{T}$, and $p, p', q, q' \in Q_\mathfrak{C}$:

  (W) *Simulate writing of the letter a into the queue*:
  $((s,p,q), a, \pi_\$(a), (s',p,q')) \in \Delta_\mathfrak{P}$ if $(s,a,s') \in \Delta_\mathfrak{T}$ and $(q,a,q') \in \Delta_\mathfrak{C}$.

  (R) *Simulate reading of the letter a from the queue*:
  $((s,p,q), \overline{a}, \pi_{\overline{\$}}(\overline{a}), (s',p',q)) \in \Delta_\mathfrak{P}$ if $(s,\overline{a},s') \in \Delta_\mathfrak{T}$ and $(p,a,p') \in \Delta_\mathfrak{C}$.

  In other words, we have the following four cases:

(1) $((s,p,q),\$^n) \xrightarrow{a}_\mathfrak{P} ((s',p,q'),\$^n)$ if, and only if, $a \in A \setminus \{\$\}$, $s \xrightarrow{a}_\mathfrak{T} s'$, and $q \xrightarrow{a}_\mathfrak{C} q'$.

(2) $((s,p,q),\$^n) \xrightarrow{\$}_\mathfrak{P} ((s',p,q'),\$^{n+1})$ if, and only if, $s \xrightarrow{\$}_\mathfrak{T} s'$ and $q \xrightarrow{\$}_\mathfrak{C} q'$.

(3) $((s,p,q),\$^n) \xrightarrow{\overline{a}}_\mathfrak{P} ((s',p',q),\$^n)$ if, and only if, $a \in A \setminus \{\$\}$, $s \xrightarrow{\overline{a}}_\mathfrak{T} s'$, and $p \xrightarrow{a}_\mathfrak{C} p'$.

(4) $((s,p,q),\$^n) \xrightarrow{\overline{\$}}_\mathfrak{P} ((s',p',q),\$^{n-1})$ if, and only if, $n > 0$, $s \xrightarrow{\overline{\$}}_\mathfrak{T} s'$, and $p \xrightarrow{\$}_\mathfrak{C} p'$.

Now, we assign to the configuration $((s,p,q),\$^n)$ the set of all words being the labeling of a run from p to q in $\mathfrak{C}$ and containing n appearances of the letter \$ (which marks the beginning of each word from W). Formally, our assignment is the mapping $[\![.]\!]_\mathfrak{P}\colon \mathrm{Conf}_\mathfrak{P} \to 2^{A^*}$ with

$$[\![((s,p,q),\$^n)]\!]_\mathfrak{P} := L(\mathfrak{C}_{p\to q}) \cap \pi_\$^{-1}(\$^n)$$

for each $s \in Q_{\mathfrak{T}}$, $p, q \in Q_{\mathfrak{C}}$, and $n \in \mathbb{N}$. This language represents the contents of all configurations (s, v) whose abstraction (as explained above) is (s, p, q, n).

Next, we state that the set of reachable queue contents coincides with the semantics of the reachable, accepting configurations of the PDA $\mathfrak{P}$.

Proposition 10.2.13. *We have* $\textsc{Reach}(L, (W\overline{R})^*) = \bigcup_{\sigma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}} [\![\sigma]\!]_{\mathfrak{P}}$.

With the help of Proposition 10.2.13 we are able to prove Theorem 10.2.12. So, we will first prove this theorem and afterwards we show the correctness of the PDA $\mathfrak{P}$ and its semantics.

Proof (of Theorem 10.2.12). Due to Theorem 9.2.1 we can compute a $\mathfrak{P}$-NFA accepting the configurations from $\mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}})$ in polynomial time. Let $\mathfrak{A} = (Q_{\mathfrak{A}}, \{\$\}, I_{\mathfrak{A}}, \Delta_{\mathfrak{A}}, F_{\mathfrak{A}})$ be this $\mathfrak{P}$-NFA with $C(\mathfrak{A}) = \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}})$. Then, we have $\$^n \in L(\mathfrak{A}_{(s,p,q) \to F_{\mathfrak{A}}})$ if, and only if, $((s, p, q), \$^n) \in \mathrm{Conf}_{\mathfrak{P}}$ is reachable from an initial configuration of $\mathfrak{P}$.

The following language is regular as well:

$$K := \bigcup_{(s,p,q) \in F_{\mathfrak{P}}} \Big(L(\mathfrak{C}_{p \to q}) \cap \pi_{\$}^{-1}\big(L(\mathfrak{A}_{(s,p,q) \to F_{\mathfrak{A}}})\big)\Big).$$

Later we will see $K = \textsc{Reach}(L, (W\overline{R})^*)$. But before, we want to give an intuition of the definition of K. This language contains all words $v \in A^*$ such that:

- v is the label of a run in $\mathfrak{C}$ from p to q, where q is accepting in $\mathfrak{C}$ (note that p is not necessarily initial) and
- there is a number $n \in \mathbb{N}$ such that v contains n \$'s and $\$^n \in L(\mathfrak{A}_{(s,p,q) \to F_{\mathfrak{A}}})$ holds. This implies that the (accepting) configuration $((s, p, q), \$^n)$ is reachable in $\mathfrak{P}$ from an initial configuration.

▷ *Claim.* *We have* $K = \textsc{Reach}(L, (W\overline{R})^*)$.

Proof. Due to Proposition 10.2.13 it suffices to show $K = \bigcup_{\sigma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}} [\![\sigma]\!]_{\mathfrak{P}}$.

First, let $v \in [\![\sigma]\!]_{\mathfrak{P}}$ for a configuration $\sigma = ((s, p, q), \$^n) \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$. Then we have $(s, p, q) \in F_{\mathfrak{P}}$ and $\$^n \in L(\mathfrak{A}_{(s,p,q) \to F_{\mathfrak{A}}})$. Hence, we have

$$\begin{aligned} v \in [\![\sigma]\!]_{\mathfrak{P}} &= L(\mathfrak{C}_{p \to q}) \cap \pi_{\$}^{-1}(\$^n) \\ &\subseteq L(\mathfrak{C}_{p \to q}) \cap \pi_{\$}^{-1}\big(L(\mathfrak{A}_{(s,p,q) \to F_{\mathfrak{A}}})\big) && \text{(since } \$^n \in L(\mathfrak{A}_{(s,p,q) \to F_{\mathfrak{A}}})\text{)} \\ &\subseteq K\,. && \text{(since } (s, p, q) \in F_{\mathfrak{P}}\text{)} \end{aligned}$$

Conversely, let $v \in K$. Then there is $(s, p, q) \in F_{\mathfrak{P}}$ with

$$v \in L(\mathfrak{C}_{p \to q}) \cap \pi_{\$}^{-1}\big(L(\mathfrak{A}_{(s,p,q) \to F_{\mathfrak{A}}})\big)\,.$$

Set $n := |v|_{\$}$. Then we have for $\sigma := ((s, p, q), \$^n) \in \mathrm{Conf}_{\mathfrak{A}}$:

$$v \in L(\mathfrak{C}_{p \to q}) \cap \pi_{\$}^{-1}(\$^n) = [\![\sigma]\!]_{\mathfrak{P}}\,.$$

Now we have to show that $\sigma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$ holds. First, we know $\$^n \in L(\mathfrak{A}_{(s,p,q) \to F_{\mathfrak{A}}})$ implying that σ is reachable from an initial configuration of $\mathfrak{P}$. Additionally, this configuration is final since $(s, p, q) \in F_{\mathfrak{P}}$ holds. In other words, we have $\sigma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$. This finishes the inclusion "$\subseteq$". ◁

Since each intermediate step of our computation is possible in polynomial time, we can compute an NFA accepting K in polynomial time as well. ◂

The Correctness of Proposition 10.2.13

Next, we want to prove the correctness of Proposition 10.2.13. We prove this with the help of two lemmas each proving one inclusion. First, we show that each reachable queue content belongs to the semantics of a reachable configuration of $\mathfrak{P}$. To this end, we consider an initial queue content $v \in L$ and an action sequence $t \in (W\overline{R})^*$ with $[\![t]\!](v) \neq \bot$. We construct a t-labeled run of $\mathfrak{P}$ such that the i^{th} intermediate result v_i on computation of $[\![t]\!](v)$ is covered by the semantics of the i^{th} configuration in our constructed run.

Concretely, we have runs $p_0, \ldots, p_\ell$ and $s_0, \ldots, s_m$ in $\mathfrak{C}$ and $\mathfrak{T}$ labeled with $v\,\mathrm{wrt}(t)$ and t, respectively, from an initial state to an accepting state. The i^{th} configuration σ_i on our run $\mathfrak{P}$ consists of s_i, two states p_{x_i} and p_{y_i} (where $0 \leq x_i \leq y_i \leq \ell$), and the number of \$'s on the subrun $p_{x_i}, \ldots, p_{y_i}$ in $\mathfrak{C}$. Then we will see that $v_i \in [\![\sigma_i]\!]_{\mathfrak{P}}$ holds, which finally implies $[\![t]\!](v) = v_m \in [\![\sigma_m]\!]_{\mathfrak{P}}$.

Example 10.2.14. Consider $L := \{\varepsilon\}$, $W := \{\$a, \$b\}$, and $R := \$^* \sqcup\!\sqcup\, b = \$^* b \$^*$. Then the languages LW^* and $(W\overline{R})^*$ are accepted by the NFAs $\mathfrak{C}$ and $\mathfrak{T}$ depicted in Figure 10.2. Let $t = \$b\overline{\$}\overline{b} \in (W\overline{R})^*$. Then we have

$$[\![\$b\overline{\$}\overline{b}]\!](\varepsilon) = [\![b\overline{\$}\overline{b}]\!](\$) = [\![\overline{\$}\overline{b}]\!](\$b) = [\![\overline{b}]\!](b) = \varepsilon \neq \bot\,.$$

Consider the accepting runs $p_1 \xrightarrow{\$}_{\mathfrak{C}} p_2 \xrightarrow{b}_{\mathfrak{C}} p_1$ and $s_1 \xrightarrow{\$}_{\mathfrak{T}} s_2 \xrightarrow{b}_{\mathfrak{T}} s_3 \xrightarrow{\overline{\$}}_{\mathfrak{T}} s_3 \xrightarrow{\overline{b}}_{\mathfrak{T}} s_4$ in $\mathfrak{C}$ and $\mathfrak{T}$ labeled with $\varepsilon\,\mathrm{wrt}(t) = \$b$ and t, respectively. Then we construct the following run in $\mathfrak{P}$:

$$\begin{aligned}\mathrm{Init}_{\mathfrak{P}} \ni ((s_1, p_1, p_1), 0) &\xrightarrow{\$}_{\mathfrak{P}} ((s_2, p_1, p_2), 1) \xrightarrow{b}_{\mathfrak{P}} ((s_3, p_1, p_1), 1)\\ &\xrightarrow{\overline{\$}}_{\mathfrak{P}} ((s_3, p_2, p_1), 0) \xrightarrow{\overline{b}}_{\mathfrak{P}} ((s_4, p_1, p_1), 0) \in \mathrm{Final}_{\mathfrak{P}}\,.\end{aligned}$$

Then we can see $[\![t]\!](\varepsilon) = \varepsilon \in [\![((s_4, p_1, p_1), 0)]\!]_{\mathfrak{P}}$. ⌟

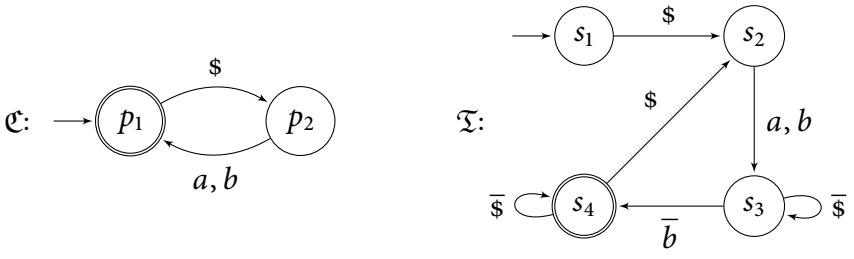

Figure 10.2. NFAs $\mathfrak{C}$ and $\mathfrak{T}$ accepting LW^* and $(W\overline{R})^*$, respectively, from Example 10.2.14.

Lemma 10.2.15. *Let $t \in (W\overline{R})^*$ and $v \in L$ with $[\![t]\!](v) \neq \bot$. Then there is a configuration $\sigma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$ with $[\![t]\!](v) \in [\![\sigma]\!]_{\mathfrak{P}}$.*

Proof. Let $t = w_1\overline{r_1}\dots w_k\overline{r_k}$ with $w_1, \dots, w_k \in W$ and $r_1, \dots, r_k \in R$. We have $vw_1 \dots w_k \in LW^* = L(\mathfrak{C})$. Hence, there is a run $p_0, \dots, p_\ell$ labeled with $vw_1 \dots w_k$ in $\mathfrak{C}$ from $p_0 \in I_\mathfrak{C}$ to $p_\ell \in F_\mathfrak{C}$. Additionally, we have $t \in (W\overline{R})^* = L(\mathfrak{T})$ and therefore a run $s_0, \dots, s_m$ in $\mathfrak{T}$ labeled with t from $s_0 \in I_\mathfrak{T}$ to $s_m \in F_\mathfrak{T}$.

By $t \in \Sigma^m$ there are basic actions $\alpha_1, \dots, \alpha_m \in \Sigma$ with $t = \alpha_1 \dots \alpha_m$. Since $[\![t]\!](v) \neq \bot$ there are intermediate queue contents $v_0, \dots, v_m \in A^*$ with $v_0 = v$ and $v_{i+1} = [\![\alpha_{i+1}]\!](v_i)$ for each $0 \leq i < m$. This implies $v_i = [\![\alpha_1 \dots \alpha_i]\!](v_0)$ and, hence, $v \operatorname{wrt}(\alpha_1 \dots \alpha_i) = \operatorname{rd}(\alpha_1 \dots \alpha_i)\, v_i$ by Observation 3.2.2. Since $v \operatorname{wrt}(\alpha_1 \dots \alpha_i)$ is a prefix of $v \operatorname{wrt}(\alpha_1 \dots \alpha_m) = vw_1 \dots w_k$ we infer that v_i is a factor of the word $vw_1 \dots w_k$. Therefore, v_i is the labeling of a fragment of the run $p_0, \dots, p_\ell$ in $\mathfrak{C}$.

Now, we want to construct a run $\sigma_0, \dots, \sigma_m$ in $\mathfrak{P}$ from an initial configuration to a final configuration with labeling t. To this end, we define

- $x_i := |\operatorname{rd}(\alpha_1 \dots \alpha_i)| = |\alpha_1 \dots \alpha_i|_{\overline{A}}$,
- $y_i := |v| + |\operatorname{wrt}(\alpha_1 \dots \alpha_i)| = |v| + |\alpha_1 \dots \alpha_i|_A$, and
- $n_i := |v_i|_\$ \geq 0$.

By definition we have $0 \leq x_i \leq |\operatorname{rd}(t)| \leq m$ and $|v| \leq y_i \leq |v \operatorname{wrt}(t)| = \ell$ for each $0 \leq i \leq m$. Set $\sigma_i := ((s_i, p_{x_i}, p_{y_i}), \$^{n_i}) \in \operatorname{Conf}_\mathfrak{P}$ for each $0 \leq i \leq m$. We will prove later that $\sigma_0, \dots, \sigma_m$ is a run of $\mathfrak{P}$ with labeling t from $\operatorname{Init}_\mathfrak{P}$ to $\operatorname{Final}_\mathfrak{P}$. But first, we have to show the following statement:

▷ *Claim 1.* *For each $0 \leq i \leq m$ we have $n_i = |\alpha_1 \dots \alpha_i|_\$ - |\alpha_1 \dots \alpha_i|_{\overline{\$}}$.*

Proof. We show this by induction on i. The case $i = 0$ is obvious since $n_0 = 0$ by $v_0 = v \in L \subseteq (A \setminus \{\$\})^*$ and $\alpha_1 \dots \alpha_0 = \varepsilon$. Now, let $i \geq 0$. The induction hypothesis holds for i and we prove the equation for $i + 1$. Then we have to distinguish two cases:

(1) $\alpha_{i+1} = a \in A$. Then we have $v_{i+1} = [\![a]\!](v_i) = v_i a$ and therefore

$$\begin{aligned} n_{i+1} = |v_{i+1}|_\$ = |v_i|_\$ + |a|_\$ = n_i + |a|_\$ &\overset{\text{i.h.}}{=} |\alpha_1 \dots \alpha_i|_\$ - |\alpha_1 \dots \alpha_i|_{\overline{\$}} + |a|_\$ \\ &= |\alpha_1 \dots \alpha_i a|_\$ - |\alpha_1 \dots \alpha_i a|_{\overline{\$}} = |\alpha_1 \dots \alpha_{i+1}|_\$ - |\alpha_1 \dots \alpha_{i+1}|_{\overline{\$}}\,. \end{aligned}$$

(2) $\alpha_{i+1} = \overline{a} \in \overline{A}$. Then by $[\![\overline{a}]\!](v_i) = v_{i+1} \neq \bot$ we have $v_i = av_{i+1}$. Hence, we have

$$\begin{aligned} n_{i+1} = |v_{i+1}|_\$ = |v_i|_\$ - |\overline{a}|_{\overline{\$}} = n_i - |\overline{a}|_{\overline{\$}} &\overset{\text{i.h.}}{=} |\alpha_1 \dots \alpha_i|_\$ - |\alpha_1 \dots \alpha_i|_{\overline{\$}} - |\overline{a}|_{\overline{\$}} \\ &= |\alpha_1 \dots \alpha_i \overline{a}|_\$ - |\alpha_1 \dots \alpha_i \overline{a}|_{\overline{\$}} = |\alpha_1 \dots \alpha_{i+1}|_\$ - |\alpha_1 \dots \alpha_{i+1}|_{\overline{\$}}\,. \end{aligned}$$

◁

Next, we prove that our run in $\mathfrak{P}$ starts in an initial configuration:

▷ *Claim 2.* *We have $\sigma_0 \in \operatorname{Init}_\mathfrak{P}$ and $v = v_0 \in [\![\sigma_0]\!]_\mathfrak{P}$.*

Proof. We have $x_0 = n_0 = 0$ and $y_0 = |v|$. Due to the choice of the run $p_0, \dots, p_\ell$ we have $p_0 \in I_\mathfrak{C}$ and $p_0 \xrightarrow{v}_\mathfrak{C} p_{|v|}$ and therefore $p_{|v|} \in \{q \in Q_\mathfrak{C} \mid \exists u \in L \colon I_\mathfrak{C} \xrightarrow{u}_\mathfrak{C} q\} = Q_L$. Additionally, by the choice of $s_0, \dots, s_m$ we have $s_0 \in I_\mathfrak{T}$. Hence, we have $\sigma_0 = ((s_0, p_0, p_{|v|}), \varepsilon) \in \operatorname{Init}_\mathfrak{P}$. By $v \in L \subseteq (A \setminus \{\$\})^*$ we can also infer $v \in L(\mathfrak{C}_{p_0 \to p_{|v|}}) \cap \pi_\$^{-1}(\$^0) = [\![\sigma_0]\!]_\mathfrak{P}$. ◁

Now, we show that $\sigma_0, \dots, \sigma_m$ is a run in $\mathfrak{P}$ labeled with t such that the semantics of σ_i contains v_i:

▷ *Claim 3.* *For each $0 \leq i < m$ we have $\sigma_i \xrightarrow{\alpha_{i+1}}_\mathfrak{P} \sigma_{i+1}$ and $v_{i+1} \in [\![\sigma_{i+1}]\!]_\mathfrak{P}$.*

Proof. We prove this statement by induction on i. By Claim 2 we know that $v_0 \in [\![\sigma_0]\!]_{\mathfrak{P}}$. Hence, we only have to prove the induction step. To this end, let $i \geq 0$. By the choice of the run $s_0, \ldots, s_m$ in $\mathfrak{T}$ we have $(s_i, \alpha_{i+1}, s_{i+1}) \in \Delta_{\mathfrak{T}}$. Now, we have to distinguish two cases:

(W) $\alpha_{i+1} = a \in A$. Then we have $x_{i+1} = x_i$, $y_{i+1} = y_i + 1$, and $n_{i+1} = n_i + |a|_{\$}$ (by Claim 1). By the choice of the run $p_0, \ldots, p_\ell$ there is an edge $(p_{y_i}, a, p_{y_{i+1}}) \in \Delta_{\mathfrak{C}}$. Hence, there is a transition

$$((s_i, p_{x_i}, p_{y_i}), a, \pi_{\$}(a), (s_{i+1}, p_{x_i}, p_{y_{i+1}})) \in \Delta_{\mathfrak{P}}$$

and, therefore, $\sigma_i \xrightarrow{a}_{\mathfrak{P}} \sigma_{i+1}$. Furthermore, we have by induction hypothesis $v_i \in [\![\sigma_i]\!]_{\mathfrak{P}}$ implying

$$\begin{aligned} v_{i+1} = [\![v_i]\!](a) = v_i a &\in [\![\sigma_i]\!]_{\mathfrak{P}} \cdot a \\ &= \left(L(\mathfrak{C}_{p_{x_i} \to p_{y_i}}) \cap \pi_{\$}^{-1}(\$^{n_i})\right) \cdot a \\ &\subseteq \left(L(\mathfrak{C}_{p_{x_i} \to p_{y_i}}) \cap \pi_{\$}^{-1}(\$^{n_i})\right) \cdot \left(L(\mathfrak{C}_{p_{y_i} \to p_{y_{i+1}}}) \cap \pi_{\$}^{-1}(\$^{|a|_{\$}})\right) \\ &\subseteq \left(L(\mathfrak{C}_{p_{x_i} \to p_{y_i}}) L(\mathfrak{C}_{p_{y_i} \to p_{y_{i+1}}})\right) \cap \pi_{\$}^{-1}(\$^{n_i}\$^{|a|_{\$}}) \\ &\subseteq L(\mathfrak{C}_{p_{x_i} \to p_{y_{i+1}}}) \cap \pi_{\$}^{-1}(\$^{n_i + |a|_{\$}}) = [\![\sigma_{i+1}]\!]_{\mathfrak{P}} . \end{aligned}$$

(R) $\alpha_{i+1} = \overline{a} \in \overline{A}$. Here, we have $x_{i+1} = x_i + 1$, $y_{i+1} = y_i$, and $n_{i+1} = n_i - |\overline{a}|_{\overline{\$}} \geq 0$ (by Claim 1). Due to $\bot \neq v_{i+1} = [\![\overline{a}]\!](v_i)$ we have $v_i = a v_{i+1}$. Since $p_{x_i}, \ldots, p_{y_i}$ is a run labeled with v_i and a is a prefix of v_i, this run begins with an a-edge. Hence, we have $(p_{x_i}, a, p_{x_{i+1}}) \in \Delta_{\mathfrak{C}}$ and, therefore,

$$((s_i, p_{x_i}, p_{y_i}), \overline{a}, \pi_{\overline{\$}}(\overline{a}), (s_{i+1}, p_{x_{i+1}}, p_{y_i})) \in \Delta_{\mathfrak{P}}$$

implying $\sigma_i \xrightarrow{\overline{a}}_{\mathfrak{P}} \sigma_{i+1}$. By the induction hypothesis we have $v_i \in [\![\sigma_i]\!]_{\mathfrak{P}} = L(\mathfrak{C}_{p_{x_i} \to p_{y_i}}) \cap \pi_{\$}^{-1}(\$^{n_i})$. Since $v_i = a v_{i+1}$ and $(p_{x_i}, a, p_{x_{i+1}}) \in \Delta_{\mathfrak{C}}$ we know that

$$a v_{i+1} = v_i \in a \cdot \left(L(\mathfrak{C}_{p_{x_{i+1}} \to p_{y_i}}) \cap \pi_{\$}^{-1}(\$^{n_i - |\overline{a}|_{\overline{\$}}})\right) = a \cdot [\![\sigma_{i+1}]\!]_{\mathfrak{P}}$$

which implies $v_{i+1} = {}_a\backslash v_i \in [\![\sigma_{i+1}]\!]_{\mathfrak{P}}$. ◁

Finally, we have to show that our run in $\mathfrak{P}$ ends up in an accepting configuration:

▷ *Claim 4. We have $\sigma_m \in \mathrm{Final}_{\mathfrak{P}}$.*

Proof. We have $y_\ell = |v| + |\mathrm{wrt}(\alpha_1 \ldots \alpha_\ell)| = |v\,\mathrm{wrt}(t)| = \ell$ implying $p_{y_m} = p_\ell \in F_{\mathfrak{C}}$. Additionally, by the choice of our run in $\mathfrak{T}$ we have $s_m \in F_{\mathfrak{T}}$. Hence, $\sigma_m = ((s_m, p_{x_m}, p_\ell), \$^{n_m}) \in \mathrm{Final}_{\mathfrak{P}}$. ◁

All in all, from the Claims 2 to 4 we obtain $[\![t]\!](v) = v_m \in [\![\sigma_m]\!]_{\mathfrak{P}}$ and σ_m is an accepting configuration which is reachable from an initial configuration. ◂

From Lemma 10.2.15 we learn the inclusion "$\subseteq$" of Proposition 10.2.13, i.e.,

$$\textsc{Reach}(L, (W\overline{R})^*) \subseteq \bigcup_{\sigma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}} [\![\sigma]\!]_{\mathfrak{P}} .$$

Recall that we have proven this by combining a run in $\mathfrak{C}$ labeled with $v\,\mathrm{wrt}(t)$ with another run in $\mathfrak{T}$ labeled with t. The result was a new run of $\mathfrak{P}$ labeled with t simulating the computation $[\![t]\!](v) \neq \bot$.

Now, we want to prove the converse implication. A first approach to prove this statement would be the following: let $\sigma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$ and $w \in [\![\sigma]\!]_{\mathfrak{P}}$. Then there is a run in $\mathfrak{P}$ from $\mathrm{Init}_{\mathfrak{P}}$ to σ labeled with $t \in L(\mathfrak{T}) = (W\overline{R})^*$. Unfortunately, it is possible that $[\![t]\!](v) \neq w$ holds for any initial queue content $v \in L$ as the following example proves:

Example 10.2.16. We continue Example 10.2.14. Consider the following accepting run of $\mathfrak{P}$:

$$\begin{aligned}\mathrm{Init}_{\mathfrak{P}} \ni ((s_1,p_1,p_1),0) &\xrightarrow{\$}_{\mathfrak{P}} ((s_2,p_1,p_2),1) \xrightarrow{a}_{\mathfrak{P}} ((s_3,p_1,p_1),1)\\ &\xrightarrow{\overline{\$}}_{\mathfrak{P}} ((s_3,p_2,p_1),0) \xrightarrow{\overline{b}}_{\mathfrak{P}} ((s_4,p_1,p_1),0) =: \sigma \in \mathrm{Final}_{\mathfrak{P}}\,.\end{aligned}$$

Then we have $[\![\sigma]\!]_{\mathfrak{P}} = \{\varepsilon\}$ and, indeed, $\varepsilon \in \textsc{Reach}(L,(W\overline{R})^*)$. However, $t = \$a\overline{\$}\overline{b}$ is no valid computation of our queue system since

$$[\![\$a\overline{\$}\overline{b}]\!](\varepsilon) = [\![a\overline{\$}\overline{b}]\!](\$) = [\![\overline{\$}\overline{b}]\!](\$a) = [\![\overline{b}]\!](a) = \bot\,.$$

The reason of this problem is the lack of memory of our pushdown automaton $\mathfrak{P}$ caused by the abstraction of the configurations. Recall that the PDA $\mathfrak{P}$ stores neither the whole content of the queue nor the applied action sequence - it only stores three states and the number of \$'s. This allows that the subsequences of write and read actions the PDA applies do not match (recall that, if $[\![t]\!](v) \neq \bot$ holds, $\mathrm{rd}(t)$ is a prefix of $v\,\mathrm{wrt}(t)$ according to Observation 3.2.2). However, we can avoid this problem by a modification of the write actions in our run labeled with t. Since the application of a read action in $\mathfrak{P}$ always requires a step in $\mathfrak{C}$, we can obtain an action sequence $t' \in (W\overline{R})^*$ in which we write the letters that will be read from the queue afterwards. This will finally result in $w = [\![t']\!](v) \in \textsc{Reach}(L,(W\overline{R})^*)$.

Lemma 10.2.17. *Let $\sigma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$. Then we have*

$$[\![\sigma]\!]_{\mathfrak{P}} \subseteq \textsc{Reach}(L,(W\overline{R})^*)\,.$$

Proof. Let $\sigma = ((s,p,q),\$^n) \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$ and $v \in [\![\sigma]\!]_{\mathfrak{P}}$. Since $v \in [\![\sigma]\!]_{\mathfrak{P}} = L(\mathfrak{C}_{p\to q}) \cap \pi_{\$}^{-1}(\$^n)$, $L(\mathfrak{C}) = LW^*$, and $q \in F_{\mathfrak{C}}$ there are a suffix y of a word from $L \cup W$ and $\$z_1,\ldots,\$z_n \in W$ such that $v = y\$z_1\$z_2\ldots\$z_n$. Hence, we have $y \in L(\mathfrak{C}_{p\to f_{\mathfrak{C}}}) \cap (A \setminus \{\$\})^*$. Furthermore, there is a word $t \in \Sigma^*$ which is the labeling of a run from $\mathrm{Init}_{\mathfrak{P}}$ to σ in $\mathfrak{P}$. Since every transition of the PDA $\mathfrak{P}$ simulates a transition of the NFA $\mathfrak{T}$, we obtain $t \in L(\mathfrak{T}) = (W\overline{R})^*$. Hence, there are $k \geq 0$, $\$w_1,\ldots,\$w_k \in W$, and $r_1,\ldots,r_k \in R$ with $t = \$w_1\overline{r_1}\$w_2\overline{r_2}\ldots\$w_k\overline{r_k}$.

The PDA $\mathfrak{P}$ lacks a memory of the concrete runs in $\mathfrak{C}$ and $\mathfrak{T}$ and, hence, lacks a memory of the letters that our queue system has written into the queue before. Therefore, it is possible that the transformation $[\![t]\!]$ cannot be applied to any initial queue content in L (i.e., $[\![t]\!](L) = \{\bot\}$). But due to this lack of memory we can obtain another run of $\mathfrak{P}$ from $\mathrm{Init}_{\mathfrak{P}}$ to σ which is labeled by t from which we have replaced the infixes $\$w_1,\ldots,\$w_k$ by other write sequences from W. In other words, we construct a new action sequence $t' \in L(\mathfrak{T}) = (W\overline{R})^*$ with $\mathrm{rd}(t') = \mathrm{rd}(t)$. This sequence t' is the labeling of another run in $\mathfrak{P}$ from $\mathrm{Init}_{\mathfrak{P}}$ to σ which corresponds to a valid computation of the queue system resulting in v (i.e., we have $v \in [\![t']\!](L)$).

Recall that \$-transitions in $\mathfrak{P}$ push another \$ to the stack (i.e., the number of \$'s increases) and $\overline{\$}$-transitions in $\mathfrak{P}$ remove one from the stack (i.e., the number of \$'s decreases). Therefore, since t is the labeling of a run in $\mathfrak{P}$ starting with an empty stack, each prefix of t contains at least as many \$'s as $\overline{\$}$'s. Hence, we have $|r_1\ldots r_i|_{\$} \leq |\$w_1\ldots\$w_i|_{\$} = i$ for each $1 \leq i \leq k$.

Due to $r_1, \dots, r_k \in R \subseteq A^*$ there are a number $\ell \in \mathbb{N}$ and words $x_0, \dots, x_\ell \in (A \setminus \{\$\})^*$ with $r_1 \dots r_k = x_0\$x_1\$ \dots \$x_\ell$. Hence, since $|r_1 \dots r_i|_\$ \leq i$ we know that $r_1 \dots r_i$ is a prefix of $x_0\$ \dots \$x_i$ for each $1 \leq i \leq \ell$. In particular, we have $k = |t|_\$$ and $\ell = |t|_{\overline{\$}}$ implying $n = k - \ell$ (recall that n is the number of \$'s in $v \in [\![\sigma]\!]_{\mathfrak{P}}$).

Now, we have to distinguish two cases: $\ell = 0$ and $\ell > 0$. We consider these two cases in the following two claims.

▷ *Claim 1.* $\ell = 0$ *implies* $v = y\$z_1\$z_2 \dots \$z_n = [\![\$z_1\overline{r_1}\$ \dots \$z_n\overline{r_n}]\!](x_0y) \in \textsc{Reach}(L, (W\overline{R})^*)$.

Proof. From $\ell = 0$ we infer $k = n$ and $r_1 \dots r_k = x_0 \in (A \setminus \{\$\})^*$. Therefore, a run in $\mathfrak{P}$ with labeling t from $\mathrm{Init}_{\mathfrak{P}}$ to σ requires a \$-free run in $\mathfrak{C}$ labeled with $r_1 \dots r_k$ from $I_{\mathfrak{C}}$ to $p \in Q_{\mathfrak{C}}$ (this path is represented in the second component of $\mathfrak{P}$'s control states). Due to $L(\mathfrak{C}) = LW^*$, $L \subseteq (A \setminus \{\$\})^*$, and $W \subseteq \$(A \setminus \{\$\})^*$, the word $r_1 \dots r_k$ is a prefix of a word from L. Hence, we have $x_0 = r_1 \dots r_k \in L(\mathfrak{C}_{I_{\mathfrak{C}} \to p}) \cap (A \setminus \{\$\})^*$ and $y \in L(\mathfrak{C}_{p \to f_{\mathfrak{C}}}) \cap (A \setminus \{\$\})^*$ implying $x_0y \in LW^* \cap (A \setminus \{\$\})^* = L$. Then we obtain the following equations:

$$\begin{aligned}
[\![\$z_1\overline{r_1}\$z_2\overline{r_2} \dots \$z_k\overline{r_k}]\!](x_0y) &= [\![\$z_1\overline{r_1}\$z_2\overline{r_2} \dots \$z_k\overline{r_k}]\!](r_1r_2 \dots r_ky) \\
&= [\![\$z_2\overline{r_2} \dots \$z_k\overline{r_k}]\!](r_2 \dots r_ky\$z_1) \\
&\vdots \\
&= [\![\$z_{i+1}\overline{r_{i+1}} \dots \$z_k\overline{r_k}]\!](r_{i+1} \dots r_ky\$z_1\$z_2 \dots \$z_i) \\
&\vdots \\
&= y\$z_1\$z_2 \dots \$z_k = v\,.
\end{aligned}$$

Since $\$z_1, \dots, \$z_k \in W$ and $r_1, \dots, r_k \in R$ we have $\$z_1\overline{r_1} \dots \$z_k\overline{r_k} \in (W\overline{R})^*$. Then from $x_0y \in L$ we can infer $v \in \textsc{Reach}(L, (W\overline{R})^*)$. ◁

▷ *Claim 2.* *If* $\ell > 0$ *we have*

$$v = y\$z_1\$z_2 \dots \$z_n = [\![\$x_1\overline{r_1}\$x_2\overline{r_2} \dots \$x_\ell y\overline{r_\ell}\$z_1\overline{r_{\ell+1}} \dots \$z_n\overline{r_k}]\!](x_0) \in \textsc{Reach}(L, (W\overline{R})^*)\,.$$

Proof. Since t is the labeling of a run from $\mathrm{Init}_{\mathfrak{P}}$ to σ in $\mathfrak{P}$ we can prove (by observing the second component of $\mathfrak{P}$'s control state) that there is a run in $\mathfrak{C}$ from $I_{\mathfrak{C}}$ to p labeled with $r_1 \dots r_k = x_0\$ \dots \$x_\ell$. By definition of $L(\mathfrak{C}) = LW^*$, $L \subseteq (A \setminus \{\$\})^*$, and $W \subseteq \$(A \setminus \{\$\})^*$ we have $x_0 \in L$, $\$x_1, \dots, \$x_{\ell-1} \in W$, and $\$x_\ell$ is a prefix of a word in W.

Since $x_0\$ \dots \$x_\ell \in L(\mathfrak{C}_{I_{\mathfrak{C}} \to p})$ and $v = y\$z_1\$z_2 \dots \$z_n \in L(\mathfrak{C}_{p \to F_{\mathfrak{C}}}) \subseteq [\![\sigma]\!]_{\mathfrak{P}}$ we have

$$x_0\$ \dots \$x_\ell y\$z_1\$ \dots \$z_n \in L(\mathfrak{C}) = LW^*$$

implying $\$x_\ell y \in W$. From $\$x_1, \dots, \$x_{\ell-1}, \$x_\ell y, \$z_1, \dots, \$z_n \in W$, $r_1, \dots, r_k \in R$, and $n = k - \ell$ we can also infer

$$\$x_1\overline{r_1}\$x_2\overline{r_2} \dots \$x_\ell y\overline{r_\ell}\$z_1\overline{r_{\ell+1}} \dots \$z_n\overline{r_k} \in (W\overline{R})^*\,.$$

Now, we prove $v = [\![\$x_1\overline{r_1}\$x_2\overline{r_2} \dots \$x_\ell y\overline{r_\ell}\$z_1\overline{r_{\ell+1}} \dots \$z_n\overline{r_k}]\!](x_0)$. This will finish our proof since $x_0 \in L$ holds. We verify this equation in two steps. First, we prove by induction on $1 \leq i < \ell$ that

$$[\![\$x_1\overline{r_1}\$x_2\overline{r_2} \dots \$x_i\overline{r_i}]\!](x_0) = {}_{r_1r_2 \dots r_i}\backslash(x_0\$x_1\$x_2 \dots \$x_i) \tag{10.1}$$

holds. To this end, let $i = 1$. Then we have

$$[\![\$x_1\overline{r_1}]\!](x_0) = [\![\overline{r_1}]\!](x_0\$x_1) = {}_{r_1}\backslash(x_0\$x_1)$$

which is defined since r_1 is a prefix of $x_0\$x_1$ as mentioned above. Now, let $1 < i < \ell$. Then we have

$$\begin{aligned}
[\![\$x_1\overline{r_1}\$x_2\overline{r_2}\dots\$x_i\overline{r_i}]\!](x_0) &= [\![\$x_i\overline{r_i}]\!]\left(_{r_1r_2\dots r_{i-1}}\backslash(x_0\$x_1\$x_2\dots\$x_{i-1})\right) && \text{(by i.h.)}\\
&= [\![\overline{r_i}]\!]\left(_{r_1r_2\dots r_{i-1}}\backslash(x_0\$x_1\$x_2\dots\$x_{i-1})\cdot\$x_i\right)\\
&= [\![\overline{r_i}]\!]\left(_{r_1r_2\dots r_{i-1}}\backslash(x_0\$x_1\$x_2\dots\$x_{i-1}\$x_i)\right)\\
&= {}_{r_i}\backslash\left(_{r_1r_2\dots r_{i-1}}\backslash(x_0\$x_1\$x_2\dots\$x_i)\right)\\
&= {}_{r_1r_2\dots r_i}\backslash(x_0\$x_1\$x_2\dots\$x_i)\,.
\end{aligned}$$

The last two equations hold since $r_1\dots r_i$ is a prefix of $x_0\$\dots\$x_i$ as we have mentioned above. Next, we can prove the following equations:

$$\begin{aligned}
&[\![\$x_1\overline{r_1}\$x_2\overline{r_2}\dots\$x_\ell y\overline{r_\ell}\$z_1\overline{r_{\ell+1}}\dots\$z_n\overline{r_k}]\!](x_0)\\
&= [\![\$x_\ell y\overline{r_\ell}\$z_1\overline{r_{\ell+1}}\dots\$z_n\overline{r_k}]\!]\left(_{r_1r_2\dots r_{\ell-1}}\backslash(x_0\$x_1\$x_2\dots\$x_{\ell-1})\right) && \text{(by Equation (10.1))}\\
&= [\![\overline{r_\ell}\$z_1\overline{r_{\ell+1}}\dots\$z_n\overline{r_k}]\!]\left(_{r_1r_2\dots r_{\ell-1}}\backslash(x_0\$x_1\$x_2\dots\$x_{\ell-1}\$x_\ell y)\right)\\
&= [\![\overline{r_\ell}\$z_1\overline{r_{\ell+1}}\dots\$z_n\overline{r_k}]\!]\left(_{r_1r_2\dots r_{\ell-1}}\backslash(r_1\dots r_k y)\right)\\
&= [\![\overline{r_\ell}\$z_1\overline{r_{\ell+1}}\dots\$z_n\overline{r_k}]\!](r_\ell r_{\ell+1}\dots r_k y)\\
&= [\![\$z_1\overline{r_{\ell+1}}\dots\$z_n\overline{r_k}]\!](r_{\ell+1}\dots r_k y)\\
&= [\![\$z_2\overline{r_{\ell+2}}\dots\$z_n\overline{r_k}]\!](r_{\ell+2}\dots r_k y\$z_1)\\
&\vdots\\
&= y\$z_1\$\dots\$z_n = v\,.
\end{aligned}$$

◁

Hence, $v \in \textsc{Reach}(L, (W\overline{R})^*)$ holds in any case. Since $v \in [\![\sigma]\!]_{\mathfrak{P}}$ was arbitrary, we obtain $[\![\sigma]\!]_{\mathfrak{P}} \subseteq \textsc{Reach}(L, (W\overline{R})^*)$. ◂

The Main Result

Until now we have seen the effective preservation of regularity if the read-write independent language $T \subseteq \Sigma^*$ satisfies a special condition, namely, $T = W\overline{R}$ where $W \subseteq \$(A \setminus \{\$\})^*$ and $R = \$^* \sqcup\!\sqcup R$. We will use this very special case to obtain our main theorem: the map $L \mapsto \textsc{Reach}(L, T^*)$ effectively preserves regularity for arbitrary regular, read-write independent languages $T \subseteq \Sigma^*$.

Theorem 10.2.18. *Let A be an alphabet, $L \subseteq A^*$ be regular, and $T \subseteq \Sigma^*$ be read-write independent and regular. Then $\textsc{Reach}(L, T^*)$ and $\textsc{BackReach}(L, T^*)$ are regular. In particular, we can compute NFAs accepting $\textsc{Reach}(L, T^*)$ and $\textsc{BackReach}(L, T^*)$, respectively, from NFAs accepting L and T in polynomial time.*

We first consider the effective and even efficient regularity of $\textsc{Reach}(L, T^*)$. Recall that we are able to de-shuffle T by Lemma 10.2.11, i.e., we have

$$\textsc{Reach}(L, T) = \textsc{Reach}(L, \mathrm{wrt}(T)\overline{\mathrm{rd}(T)})\,.$$

By induction on the number of iterations we can also prove

$$\textsc{Reach}(L, T^*) = \textsc{Reach}(L, (\mathrm{wrt}(T)\overline{\mathrm{rd}(T)})^*)\,.$$

So, when computing $\textsc{Reach}(L, T^*)$ it suffices to only consider the de-shuffled words in T. Now, we will insert the begin marker $\$ \notin A$ into the de-shuffled words. Then we compute the reachable contents of our modified queue system. Finally, using a projection we remove the $\$$'s and obtain the reachable queue contents of the original queue system.

We first show that the last step of this construction is valid, i.e., that the insertion of the begin marker $\$$ at the appropriate positions into the de-shuffled action sequences does not change the behavior of the queue system.

Lemma 10.2.19. *Let A be an alphabet, $\$ \notin A$ be another symbol, and $L, W, R \subseteq A^*$. Set $W' := \$W$ and $R' := \$^* \sqcup\!\sqcup R$. Then we have*

$$\textsc{Reach}(L, W\overline{R}) = \pi_A(\textsc{Reach}(L, W'\overline{R'}))\,.$$

Proof. First, let $x \in \textsc{Reach}(L, W\overline{R})$. Then there are $v \in L$, $w \in W$, and $r \in R$ with $[\![w\bar{r}]\!](v) = x \neq \bot$. Then by Observation 3.2.2 we have $rx = vw$. We can construct $r' \in \$^* \sqcup\!\sqcup r \subseteq \$^* \sqcup\!\sqcup R = R'$ and $x' \in \$^* \sqcup\!\sqcup x$ satisfying $r'x' = v\$w$, i.e., we have $x' = {}_{r'}\backslash v\$w$. Hence, the following holds:

$$\bot \neq x' = {}_{r'}\backslash v\$w = [\![\overline{r'}]\!](v\$w) = [\![\$w\overline{r'}]\!](v) \in \textsc{Reach}(L, W'\overline{R'})$$

implying $x = \pi_A(x') \in \pi_A(\textsc{Reach}(L, W'\overline{R'}))$.

Now, let $x \in \pi_A(\textsc{Reach}(L, W'\overline{R'}))$. Then there is $x' \in \textsc{Reach}(L, W'\overline{R'})$ with $x = \pi_A(x')$, i.e., we find $v \in L$, $w' \in W'$, and $r' \in R'$ with $[\![w'\overline{r'}]\!](v) = x' \neq \bot$. Again, application of Observation 3.2.2 yields $r'x' = vw'$. Since π_A is a homomorphism, we can infer

$$\pi_A(r')x = \pi_A(r')\pi_A(x') = \pi_A(r'x') = \pi_A(vw') = \pi_A(v)\pi_A(w') = v\pi_A(w')\,.$$

Therefore, we obtain $x = {}_{\pi_A(r)}\backslash v\pi_A(w')$ implying

$$\bot \neq x = {}_{\pi_A(r')}\backslash v\pi_A(w') = [\![\pi_A(w')\overline{\pi_A(r')}]\!](v)\,.$$

Since we know $\pi_A(w) \in \pi_A(W') = W$ and $\pi_A(r') \in \pi_A(R') = R$, we can finally infer $x \in \textsc{Reach}(L, W\overline{R})$. ◂

By iterated application of Lemma 10.2.19 we can also infer the following equation:

$$\textsc{Reach}(L, (W\overline{R})^*) = \pi_A(\textsc{Reach}(L, (W'\overline{R'})^*))\,.$$

Hence, we are now able to prove our main theorem in this section:

Proof (of Theorem 10.2.18). Let $W := \mathrm{wrt}(T)$ and $R := \mathrm{rd}(T)$ which are both regular languages by the closure properties of the class of regular languages. We introduce a begin marker $\$ \notin A$ for words in W. Then we can compute NFAs accepting $W' := \$W$ and $R' := \$^* \sqcup\!\sqcup R$. By

Theorem 10.2.12 we know that $\textsc{Reach}(L, (W'\overline{R'})^*)$ is effectively and even efficiently regular as well. Then by iterated application of the Lemmata 10.2.11 and 10.2.19 we can infer that

$$\textsc{Reach}(L, T^*) = \textsc{Reach}(L, (W\overline{R})^*) = \pi_A(\textsc{Reach}(L, (W'\overline{R'})^*))$$

holds. Hence, due to the closure properties of regular languages, $\textsc{Reach}(L, T^*)$ is effectively regular. Note that the modifications of W and R as well as the projection to A are possible in linear time and space. Hence, an NFA accepting $\textsc{Reach}(L, T^*)$ can be computed still in polynomial time.

Finally, we have to consider $\textsc{BackReach}(L, T^*)$. Due to Lemma 10.2.11 and Theorem 10.2.1 we have

$$\textsc{BackReach}(L, T^*) = \textsc{BackReach}(L, (W\overline{R})^*) = \textsc{Reach}(L^R, (R^R\overline{W^R})^*)^R.$$

By the closure properties and the statement above we obtain the effective and even efficient regularity of $\textsc{BackReach}(L, T^*)$. ◂

We can use Theorem 10.2.18 to prove the effective preservation of regularity of several further language classes. The following corollary lists some of them. In particular, we can see that the main result from Boigelot et al. in [Boi+97] follows from Theorem 10.2.18. Concretely, we will see that $L \mapsto \textsc{Reach}(L, t^*)$ preserves regularity for any action sequence $t \in \Sigma^*$.

Corollary 10.2.20. *Let A be an alphabet and $L \subseteq A^*$ and $T \subseteq \Sigma^*$ be two regular languages. Then* $\textsc{Reach}(L, T^*)$ *and* $\textsc{BackReach}(L, T^*)$ *are regular if*

(1) $T = \overline{R_1}W\overline{R_2}$ *for three regular languages* $W, R_1, R_2 \subseteq A^*$,

(2) $T = W_1\overline{R}W_2$ *for three regular languages* $W_1, W_2, R \subseteq A^*$,

(3) $T = \{t\}$ *for an action sequence* $t \in \Sigma^*$ *(cf. [Boi+97]), or*

(4) $T \subseteq A^* \cup \overline{A}^*$.

In all of these cases the computation of NFAs accepting $\textsc{Reach}(L, T^*)$ *and* $\textsc{BackReach}(L, T^*)$*, respectively, from NFAs accepting L and T is possible in polynomial time.*

Proof. First, we prove (1). Then we have

$$(\overline{R_1}W\overline{R_2})^* = \{\varepsilon\} \cup \overline{R_1}(W\overline{R_2R_1})^*W\overline{R_2}.$$

Then due to Proposition 10.2.3 and Theorem 10.2.18 $\textsc{Reach}(L, (\overline{R_1}W\overline{R_2})^*)$ is efficiently regular. The proof of (2) is very similar to the first one.

Next, we consider (3). Due to Corollary 4.7.15 we can compute a "simple" action sequence $s \in \overline{A}^*A^*\overline{A}^*$ with $s \equiv t$ (in polynomial time). Statement (1) implies the efficient regularity of $\textsc{Reach}(L, s^*)$. Hence, $\textsc{Reach}(L, t^*)$ has this property as well.

Finally, we consider (4). Let $W, R \subseteq A^*$ with $T = W \cup \overline{R}$ (note that we can compute NFAs accepting W and R from T). Then we have $T^* = (W^*\overline{R}^*)^*$. Hence, due to Theorem 10.2.18 $\textsc{Reach}(L, T^*)$ is effectively regular. ◂

As we have seen, Theorem 10.2.18 implies the efficient preservation of regularity for a large class of languages of action sequences. However, we think our result can be generalized to an even larger class of languages. Recall that $T \subseteq \Sigma^*$ is read-write independent if for each pair s, t of action sequences from T there is a particular de-shuffled combination $\mathrm{wrt}(s)\overline{\mathrm{rd}(t)}$ of these sequences in T. A possible generalization is to drop the requirement that this combination is de-shuffled:

Conjecture 10.2.21. *Let A be an alphabet, $L \subseteq A^*$ be regular, and $T \subseteq \Sigma^*$ be regular such that for each $s, t \in T$ there is $r \in T$ with $\mathrm{wrt}(r) = \mathrm{wrt}(s)$ and $\mathrm{rd}(r) = \mathrm{rd}(t)$. We conjecture that in this case $\textsc{Reach}(L, T^*)$ is effectively regular.*

The proof of Theorem 10.2.18 does not work in this case. At least the utilization of Lemma 10.2.11 where we de-shuffle the words from T, is impossible in certain cases. For example, we have

$$\textsc{Reach}(\varepsilon, (\overline{a}aa)^*) = \{\varepsilon\} \neq a^* = \textsc{Reach}(\varepsilon, (aa\overline{a})^*) .$$

However, possibly the construction of our PDA $\mathfrak{P}$ in the proof of Theorem 10.2.12 can be slightly modified to match this more general case.

10.3 Partially Lossy Queues

Until now we have only considered the reachability problem of reliable queues. Now, we also want to consider (partially) lossy queues. We know that the reachability problem of automata with a partially lossy queue is either undecidable (if the underlying alphabet is at least binary and contains at least one unforgettable letter, cf. Theorem 3.3.7) or very inefficient (if the underlying alphabet is at least binary and contains only forgettable letters, cf. [Sch02, CS08]). Hence, it is also reasonable to under-approximate the reachability problem in partially lossy queue systems with the help of meta-transformations as in [Boi+97, Abd+04]. In this section we want to study several types of such meta-transformations. Concretely, we want to generalize the results from the previous section to partially lossy queue systems.

Note that we have to distinguish the two semantics of partial lossiness in this case: the default semantics (represented by the lossy data type $\mathcal{PLQ}_{\mathcal{L}}$) and the read-lossy semantics (represented by the data type $\mathcal{Q}_{\mathcal{L}}$). However, we can see the following connection between these two semantics:

Theorem 10.3.1. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $L \subseteq A^*$, and $T \subseteq \Sigma^*$. Then we have the following statements:*

(1) $\textsc{Reach}_{\mathcal{PLQ}_{\mathcal{L}}}(L, T) = {\downarrow_{\sqsubseteq_{\mathcal{L}}}}\textsc{Reach}_{\mathcal{Q}_{\mathcal{L}}}(L, T)$

(2) $\textsc{BackReach}_{\mathcal{PLQ}_{\mathcal{L}}}(L, T) = \textsc{BackReach}_{\mathcal{Q}_{\mathcal{L}}}({\uparrow_{\sqsubseteq_{\mathcal{L}}}}L, T)$

Note that this statement is very similar to Proposition 9.3.2 stating the connection of pre and post in partially lossy stack systems with default and read-lossy semantics.

Proof. Similar to Proposition 3.3.4 we can show the following equivalence:

$$w \in [\![t]\!]_{\mathcal{PLQ}_{\mathcal{L}}}(v) \iff w \sqsubseteq_{\mathcal{L}} [\![t]\!]_{\mathcal{Q}_{\mathcal{L}}}(v) \tag{10.2}$$

(for a proof cf. [KKP18, Theorem 3.5]). We use this equivalence multiple times across this proof.

We first prove statement (1). Let $w \in \textsc{Reach}_{\mathcal{PLQ}_{\mathcal{L}}}(L, T)$. Then there are $v \in L$ and $t \in T$ with $w \in [\![t]\!]_{\mathcal{PLQ}_{\mathcal{L}}}(v)$. By (10.2) we know that $w \sqsubseteq_{\mathcal{L}} [\![t]\!]_{\mathcal{Q}_{\mathcal{L}}}(v)$ holds implying $w \in \downarrow_{\sqsubseteq_{\mathcal{L}}} \textsc{Reach}_{\mathcal{Q}_{\mathcal{L}}}(L, T)$.

Conversely, let $w \in \downarrow_{\sqsubseteq_{\mathcal{L}}} \textsc{Reach}_{\mathcal{Q}_{\mathcal{L}}}(L, T)$. Then there are $v \in L$ and $t \in T$ with $w \sqsubseteq_{\mathcal{L}} [\![t]\!]_{\mathcal{Q}_{\mathcal{L}}}(v)$ implying $w \in [\![t]\!]_{\mathcal{PLQ}_{\mathcal{L}}}(v)$ according to (10.2). Hence, we infer $w \in \textsc{Reach}_{\mathcal{PLQ}_{\mathcal{L}}}(L, T)$.

Now, we prove (2). Let $v \in \textsc{BackReach}_{\mathcal{PLQ}_{\mathcal{L}}}(L, T)$. Then there are $w \in L$ and $t \in T$ such that $w \in [\![t]\!]_{\mathcal{PLQ}_{\mathcal{L}}}(v)$. Due to (10.2) we know $w \sqsubseteq_{\mathcal{L}} [\![t]\!]_{\mathcal{Q}_{\mathcal{L}}}(v)$. Hence, we have $[\![t]\!]_{\mathcal{Q}_{\mathcal{L}}}(v) \in \uparrow_{\sqsubseteq_{\mathcal{L}}} w \subseteq \uparrow_{\sqsubseteq_{\mathcal{L}}} L$ resulting in $v \in \textsc{BackReach}_{\mathcal{Q}_{\mathcal{L}}}(\uparrow_{\sqsubseteq_{\mathcal{L}}} L, T)$.

Towards the converse inclusion, let $v \in \textsc{BackReach}_{\mathcal{Q}_{\mathcal{L}}}(\uparrow_{\sqsubseteq_{\mathcal{L}}} L, T)$, i.e., we have $[\![T]\!]_{\mathcal{Q}_{\mathcal{L}}}(v) \cap \uparrow_{\sqsubseteq_{\mathcal{L}}} L \neq \varnothing$. There is $t \in T$ with $[\![t]\!]_{\mathcal{Q}_{\mathcal{L}}}(v) \in \uparrow_{\sqsubseteq_{\mathcal{L}}} L$ implying the existence of $w \in L$ with $w \sqsubseteq_{\mathcal{L}} [\![t]\!]_{\mathcal{Q}_{\mathcal{L}}}(v)$. Then the application of (10.2) yields $w \in [\![t]\!]_{\mathcal{PLQ}_{\mathcal{L}}}(v)$ which finally implies $v \in \textsc{BackReach}_{\mathcal{PLQ}_{\mathcal{L}}}(L, T)$. ◂

Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $L \subseteq A^*$ be regular. Then the languages $\downarrow_{\sqsubseteq_{\mathcal{L}}} L$ and $\uparrow_{\sqsubseteq_{\mathcal{L}}} L$ are effectively and even efficiently regular. So, we are able to compute the forwards and backwards reachable sets of queue contents of a partially lossy queue with default semantics from their correspondences of partially lossy queues with read-lossy semantics. Hence, whenever we consider the reachability problem of a partially lossy queue system we only consider their read-lossy semantics. So from now on in this section, we use the data type $\mathcal{Q}_{\mathcal{L}}$ and omit the index $\mathcal{Q}_{\mathcal{L}}$ whenever the situation is clear.

Now, let $\mathcal{L} = (\varnothing, U)$ be a lossiness alphabet. Then we know $\mathcal{Q}_{\mathcal{L}} \cong \mathcal{Q}_A$, i.e., a partially lossy queue with lossiness alphabet $\mathcal{L}$ is reliable. Hence, we know $\textsc{Reach}_{\mathcal{Q}_{\mathcal{L}}} = \textsc{Reach}_{\mathcal{Q}_A}$ and $\textsc{BackReach}_{\mathcal{Q}_{\mathcal{L}}} = \textsc{BackReach}_{\mathcal{Q}_A}$. Due to Theorem 10.2.1 we have a strong duality between forwards and backwards reachability in this reliable case. However, this duality does not hold for arbitrary lossiness alphabets: if $\mathcal{L} = (F, U)$ is a lossiness alphabet with $a \in A$ and $F \neq \varnothing$, we have $\textsc{Reach}_{\mathcal{Q}_{\mathcal{L}}}(\{\varepsilon\}, \{a\}) = \{a\}$, which cannot be transformed into $\textsc{BackReach}_{\mathcal{Q}_{\mathcal{L}}}(\{\varepsilon\}, \{\overline{a}\}) = (F \setminus \{a\})^* a$ using the reversal operation, only. Hence, we have to consider forwards and backwards reachability separately in this case. Anyway, we will see later in this section that we can reduce forwards and backwards reachability for arbitrary partially lossy queues to the reachability problems in reliable queues.

Before we consider several meta-transformations, we have to analyze the effect of lossy reading of some letters. To this end, we recall that

$$\operatorname{redsup}_{\mathcal{L}}(w) = \{w_1 a_1 w_2 a_2 \dots w_n a_n \mid \forall 1 \leq i \leq n : w_i \in (F \setminus \{a_i\})^*\}$$

is the set of all reduced $\mathcal{L}$-superwords of $w = a_1 a_2 \dots a_n$ with $a_1, a_2, \dots, a_n \in A$. For a language $L \subseteq A^*$ we set $\operatorname{redsup}_{\mathcal{L}}(L) := \bigcup_{w \in L} \operatorname{redsup}_{\mathcal{L}}(w)$.

The following lemma proves that the map $\operatorname{redsup}_{\mathcal{L}} : 2^{A^*} \to 2^{A^*}$ efficiently preserves regularity:

Lemma 10.3.2. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $L \subseteq A^*$ be regular. Then the language* $\operatorname{redsup}_{\mathcal{L}}(L)$ *is regular. An NFA accepting* $\operatorname{redsup}_{\mathcal{L}}(L)$ *can be computed from an NFA accepting L in polynomial time.*

Proof idea. We construct an NFA $\mathfrak{B}$ accepting $\operatorname{redsup}_{\mathcal{L}}(L)^{\mathrm{R}}$ from an NFA $\mathfrak{A}$ accepting L^{R}. Since the class of regular languages is closed under reversal, we also obtain regularity of $\operatorname{redsup}_{\mathcal{L}}(L)$.

Let $\mathfrak{A} = (Q_{\mathfrak{A}}, A, I_{\mathfrak{A}}, \Delta_{\mathfrak{A}}, F_{\mathfrak{A}})$ be an NFA accepting L^{R}. Our NFA $\mathfrak{B}$ accepting $\operatorname{redsup}_{\mathcal{L}}(L)^{\mathrm{R}}$ does the following: while $\mathfrak{B}$ simulates the computation of the NFA $\mathfrak{A}$, it always stores the letter it has read on the previous step of $\mathfrak{A}$. After such simulation transition, $\mathfrak{B}$ is able to read some other forgettable letters.

Formally, we compute the following NFA $\mathfrak{B} = (Q_{\mathfrak{B}}, A, I_{\mathfrak{B}}, \Delta_{\mathfrak{B}}, F_{\mathfrak{B}})$:

- $Q_{\mathfrak{B}} := Q_{\mathfrak{A}} \times (A \cup \{\varepsilon\})$,
- $I_{\mathfrak{B}} := I_{\mathfrak{A}} \times \{\varepsilon\}$,
- $F_{\mathfrak{B}} := F_{\mathfrak{A}} \times (A \cup \{\varepsilon\})$, and
- $\Delta_{\mathfrak{B}} := \{((p,a), b, (q,b)) \mid (p,a,q) \in \Delta_{\mathfrak{A}}\} \cup \{((p,a), b, (p,a)) \mid a \neq \varepsilon, b \in F \setminus \{a\}\}$.

Finally, we can prove $L(\mathfrak{B}) = \operatorname{redsup}_{\mathcal{L}}(L)^{\mathrm{R}}$. ◂

Now we can state the connection between partially lossy computations $[\![t]\!]$ and reduced $\mathcal{L}$-superwords:

Lemma 10.3.3. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $v, w, t \in A^*$. Then we have $[\![\overline{t}]\!](v) = w$ if, and only if, there is $s \in \operatorname{redsup}_{\mathcal{L}}(t)$ with $v = sw$.*

Proof. We prove this by induction on the length of t. First, assume $t = \varepsilon$. Then we have $v = [\![\varepsilon]\!](v) = w$, $\varepsilon \in \operatorname{redsup}_{\mathcal{L}}(\varepsilon)$, and $v = \varepsilon w = w$.

Next, let $t = at'$ for an $a \in A$ and $t' \in A^*$. Assume $[\![\overline{t}]\!](v) = w$. Then we have $w = [\![\overline{t'}]\!]([\![\overline{a}]\!](v))$. By definition of $[\![\overline{a}]\!]$ there are $v_1 \in (F \setminus \{a\})^*$ and $v_2 \in A^*$ with $v = v_1 a v_2$, $[\![\overline{a}]\!](v) = v_2$, and $[\![\overline{t'}]\!](v_2) = w$. By induction hypothesis there is $s' \in \operatorname{redsup}_{\mathcal{L}}(t')$ with $v_2 = s'w$. Set $s := v_1 a s'$. Then we have $s \in \operatorname{redsup}_{\mathcal{L}}(at') = \operatorname{redsup}_{\mathcal{L}}(t)$ and $v = v_1 a v_2 = v_1 a s' w = sw$.

Conversely, let $s \in \operatorname{redsup}_{\mathcal{L}}(t)$ with $v = sw$. Then by definition there is $s_1 \in (F \setminus \{a\})^*$, $s_2 \in A^*$ with $s = s_1 a s_2$ and $s_2 \in \operatorname{redsup}_{\mathcal{L}}(t')$. By $v = sw$ there is $v_2 \in A^*$ with $v = sw = s_1 a s_2 w = s_1 a v_2$, i.e., $v_2 = s_2 w$. By induction hypothesis we have $[\![\overline{t'}]\!](v_2) = w$. We also have $[\![\overline{a}]\!](v) = v_2$ implying

$$w = [\![\overline{t'}]\!](v_2) = [\![\overline{t'}]\!]([\![\overline{a}]\!](v)) = [\![\overline{t}]\!](v)\,. \qquad \blacktriangleleft$$

Let $\mathcal{L} = (\varnothing, U)$ be a lossiness alphabet without forgettable letters, $t \in \Sigma^*$, and $v, w \in A^*$ with $[\![t]\!]_{Q_A}(v) = w \neq \bot$. According to Observation 3.2.2 $v \operatorname{wrt}(t) = \operatorname{rd}(t)\, w$ holds. With the help of Lemma 10.3.3 we can generalize this observation to arbitrary lossiness alphabets. To this end, let $\mathcal{L} = (F, U)$ now be an arbitrary lossiness alphabet, $t \in \Sigma^*$, and $v, w \in A^*$

with $[\![t]\!]_{Q_{\mathcal{L}}}(v) = w \neq \bot$. By Lemma 4.7.3 we can de-shuffle t in this case implying $w = [\![t]\!]_{Q_{\mathcal{L}}}(v) = [\![\mathrm{wrt}(t)\overline{\mathrm{rd}(t)}]\!]_{Q_{\mathcal{L}}}(v)$. Then by Lemma 10.3.3 there is a word $r \in \mathrm{redsup}_{\mathcal{L}}(\mathrm{rd}(t))$ with $v\,\mathrm{wrt}(t) = rw$.

Additionally, with the help of Lemma 10.3.3 we can also prove the following reductions from reachability in plq automata to their correspondences in reliable queue systems:

Proposition 10.3.4. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet and $L, T \subseteq A^*$. Then the following statements hold:*

(1) $\textsc{Reach}_{Q_{\mathcal{L}}}(L, T) = \textsc{Reach}_{Q_A}(L, T)$

(2) $\textsc{Reach}_{Q_{\mathcal{L}}}(L, \overline{T}) = \textsc{Reach}_{Q_A}(L, \overline{\mathrm{redsup}_{\mathcal{L}}(T)})$

(3) $\textsc{BackReach}_{Q_{\mathcal{L}}}(L, T) = \textsc{BackReach}_{Q_A}(L, T)$

(4) $\textsc{BackReach}_{Q_{\mathcal{L}}}(L, \overline{T}) = \textsc{BackReach}_{Q_A}(L, \overline{\mathrm{redsup}_{\mathcal{L}}(T)})$

Proof. First, we prove (1). By definition we know $[\![t]\!]_{Q_{\mathcal{L}}} = [\![t]\!]_{Q_A}$ for each $t \in A^*$. This implies

$$\textsc{Reach}_{Q_{\mathcal{L}}}(L, T) = [\![T]\!]_{Q_{\mathcal{L}}}(L) \setminus \{\bot\} = [\![T]\!]_{Q_A}(L) \setminus \{\bot\} = \textsc{Reach}_{Q_A}(L, T)\,.$$

To prove (2), let $w \in \textsc{Reach}_{Q_{\mathcal{L}}}(L, T)$. Then there are $v \in L$ and $t \in T$ with $[\![\overline{t}]\!]_{Q_{\mathcal{L}}}(v) = w$. By Lemma 10.3.3 there is $s \in \mathrm{redsup}_{\mathcal{L}}(t) \subseteq \mathrm{redsup}_{\mathcal{L}}(T)$ with $v = sw$. Then we have $w = [\![\overline{s}]\!]_{Q_A}(v) \in [\![\overline{\mathrm{redsup}_{\mathcal{L}}(T)}]\!]_{Q_A}(L)$ implying $w \in \textsc{Reach}_{Q_A}(L, \overline{\mathrm{redsup}_{\mathcal{L}}(T)})$.

Now, let $w \in \textsc{Reach}_{Q_A}(L, \overline{\mathrm{redsup}_{\mathcal{L}}(T)})$. Then there are $v \in L$ and $s \in \mathrm{redsup}_{\mathcal{L}}(T)$ with $[\![\overline{s}]\!]_{Q_A}(v) = w$ and, therefore, $v = sw$. By $s \in \mathrm{redsup}_{\mathcal{L}}(T)$ there is $t \in T$ with $s \in \mathrm{redsup}_{\mathcal{L}}(t)$. By Lemma 10.3.3 we obtain $w = [\![\overline{t}]\!]_{Q_{\mathcal{L}}}(v) \in [\![\overline{T}]\!]_{Q_{\mathcal{L}}}(L)$ implying $w \in \textsc{Reach}_{Q_{\mathcal{L}}}(L, T)$.

Equation (3) holds due to the following equations:

$$\begin{aligned}\textsc{BackReach}_{Q_{\mathcal{L}}}(L, T) &= \{x \in A^* \mid [\![T]\!]_{Q_{\mathcal{L}}}(x) \cap L \neq \varnothing\} \\ &= \{x \in A^* \mid [\![T]\!]_{Q_A}(x) \cap L \neq \varnothing\} = \textsc{BackReach}_{Q_A}(L, T)\,.\end{aligned}$$

We can prove (4) similar to (2). ◂

As a corollary of Proposition 10.3.4 we can obtain that the results from the previous section also hold for arbitrary partially lossy queue systems:

Theorem 10.3.5. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $L \subseteq A^*$ be regular, and $T \subseteq \Sigma^*$ be regular and closed under $\equiv$ (i.e., $T = \eta^{-1}(S)$ for a recognizable plq language $S \subseteq \mathbb{T}(Q_{\mathcal{L}})$). Then $\textsc{Reach}_{Q_{\mathcal{L}}}(L, T)$ and $\textsc{BackReach}_{Q_{\mathcal{L}}}(L, T)$ are efficiently regular.* ◂

Theorem 10.3.6. *Let $\mathcal{L} = (F, U)$ be a lossiness alphabet, $L \subseteq A^*$ be regular, and $T \subseteq \Sigma^*$ be regular. Then $\textsc{Reach}_{Q_\mathcal{L}}(L, T^*)$ and $\textsc{BackReach}_{Q_\mathcal{L}}(L, T^*)$ are efficiently regular if*

(1) T is read-write independent,

(2) $T = \overline{R_1} W \overline{R_2}$ for three regular languages $W, R_1, R_2 \subseteq A^*$,

(3) $T = W_1 \overline{R} W_2$ for three regular languages $W_1, W_2, R \subseteq A^*$,

(4) $T = \{t\}$ for an action sequence $t \in \Sigma^*$ (cf. [Abd+04, Boi+97]), or

(5) $T \subseteq A^* \cup \overline{A}^*$. ◂

10.4 Distributed Queues

Finally, we want to consider the reachability problem of automata having multiple reliable queues. With the help of such systems we are able to model distributed systems communicating through several reliable channels. Since such systems contain at least one queue, multi-queue systems are still as powerful as Turing-machines. Hence, their reachability problem is undecidable. Similar to the single-queue case, we can under-approximate this problem with the help of meta-transformations. To this end, we want to generalize the results from Section 10.2 in which we considered the reachability problem in reliable single-queue systems. Concretely, we check whether it is possible to compute asynchronous automata accepting forwards resp. backwards reachable configurations starting from a recognizable trace language $L \subseteq \mathbb{M}(\mathcal{A})$ of initial queue contents via certain languages $T \subseteq \mathbb{M}(\mathcal{E})$ of action traces (recall that we can understand the action sequences of a distributed queue system as traces, cf. Lemma 4.8.1). Concretely, we want to compute asynchronous automata accepting $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$, respectively, whenever $L \subseteq \mathbb{M}(\mathcal{A})$ is recognizable and $T \subseteq \mathbb{M}(\mathcal{E})$ is one of the following trace languages:

(1) $T \subseteq \mathbb{M}(\mathcal{A}) \cup \overline{\mathbb{M}(\mathcal{A})}$ is recognizable,

(2) T is recognizable and closed under behavioral equivalence $\equiv$ (i.e., $T = \eta^{-1}(S)$ for a recognizable distributed queue language $S \subseteq \mathbb{T}(Q_\mathcal{A})$),

(3) $T = (W\overline{R})^*$ for two recognizable languages $W, R \subseteq \mathbb{M}(\mathcal{A})$, or

(4) $T = \tau^*$ for a trace $\tau \in \mathbb{M}(\mathcal{E})$.

We will see in this section that the computation is possible if the first or second case holds. But in general, the computation in the other cases is impossible as the following example states:

Example 10.4.1. Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $a, b \in A$ be distinct letters with $a \parallel b$. Then we have $\textsc{Reach}(\{\varepsilon\}, (ab)^*) = (ab)^*$ which is not recognizable in $\mathbb{M}(\mathcal{A})$. ⌟

To circumvent this problem, we will introduce later some further restrictions to the third and fourth case listed above such that $\textsc{Reach}(L, T)$ preserves recognizability effectively. But

first, we will see that forwards and backwards reachability in distributed queue systems are dual:

Theorem 10.4.2. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $L \subseteq \mathbb{M}(\mathcal{A})$, and $T \subseteq \mathbb{M}(\mathcal{E})$. Then we have*

$$\textsc{BackReach}(L, T) = \textsc{Reach}(L^{\mathrm{R}}, \mathrm{d}(T))^{\mathrm{R}}.$$

Proof. This proof is essentially the same as in Theorem 10.2.1. ◂

We can use Theorem 10.4.2 to derive knowledge about BackReach from our results about Reach. However, we will also see some differences between forwards and backwards reachable trace languages later in this section.

Similar to the single-queue systems, distributed queue systems are Turing-complete and, hence, $\textsc{Reach}(L, T)$ can be any recursively enumerable trace language - even if L and T are somewhat simple trace languages. Thus, we have to approximate the set $\textsc{Reach}(L, T)$, for example with the help of meta-transformations.

We start to consider simple meta-transformations consisting of action traces only writing or only reading letters. The result and proof is similar to Proposition 10.2.3:

Proposition 10.4.3. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $L, T \subseteq \mathbb{M}(\mathcal{A})$. Then the following statements hold:*

(1) $\textsc{Reach}(L, T) = LT$ *and*

(2) $\textsc{Reach}(L, \overline{T}) = {}_T\backslash L$. ◂

Now, let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $L, T \subseteq \mathbb{M}(\mathcal{A})$ be accepted by the asynchronous automata $\mathfrak{L}$ and $\mathfrak{T}$, respectively. Then we can compute asynchronous automata accepting $\textsc{Reach}(L, T)$ and $\textsc{Reach}(L, \overline{T})$. Due to Appendix A it is possible to compute an asynchronous automaton accepting $\textsc{Reach}(L, \overline{T})$ which only differs from $\mathfrak{L}$ in its initial states. This construction is possible in cubic time. An asynchronous automaton for $\textsc{Reach}(L, T)$ can be computed having $O(|\mathfrak{L}|^3 \cdot |\mathfrak{T}|^3)$ many states in polynomial time.

If the trace languages L and T are accepted by NFAs $\mathfrak{L}$ and $\mathfrak{T}$ which are not asynchronous, i.e., if L and T are rational in $\mathbb{M}(\mathcal{A})$, $\textsc{Reach}(L, T)$ also is efficiently rational. This trace language is accepted by an NFA having linearly many states in the number of states of $\mathfrak{L}$ and $\mathfrak{T}$. However, there are rational trace languages L and T such that $\textsc{Reach}(L, \overline{T})$ is not even recursive:

Remark 10.4.4. Let $B = \{a, b\}$ be an alphabet and $I = (x_j, y_j)_{1 \leq j \leq k}$ be an instance of the PCP over the alphabet B. We define the following distributed alphabet $\mathcal{A} = (A, P, \mathrm{M})$:

- $A := \{a, b, c, d\}$,
- $P := \{1, 2\}$, and

- $\mathrm{M} := \{(a,1),(b,1),(c,2),(d,2)\}$.

Additionally, we define the rational trace languages L and T as follows:

$$L := \{[x_j c d^j] \mid 1 \leq j \leq k\}^+ \quad \text{and} \quad T := \{[y_j c d^j] \mid 1 \leq j \leq k\}^* .$$

Then we have $\varepsilon \in \textsc{Reach}(L, \overline{T}) = {}_T\backslash L$ if, and only if, I has a solution of the PCP, which is an undecidable problem. ⌟

Note that this negative result requires L and T to be non-recognizable. So, if L is recognizable and T is rational in $\mathbb{M}(\mathcal{A})$, the language $\textsc{Reach}(L, \overline{T}) = {}_T\backslash L$ is efficiently recognizable. If L is rational and T is recognizable, then $\textsc{Reach}(L, \overline{T})$ is effectively rational. The construction of an NFA accepting this language is possible in polynomial time if we assume that the underlying distributed alphabet $\mathcal{A}$ is fixed (cf. Appendix A). A summary of these results can be found in Figure 10.3.

$L \subseteq \mathbb{M}(\mathcal{A})$	$T \subseteq \mathbb{M}(\mathcal{E})$	$\textsc{Reach}(L, T)$	$\textsc{Reach}(L, \overline{T})$
recognizable	recognizable	recognizable	recognizable
recognizable	rational	rational	recognizable
rational	recognizable	rational	rational
rational	rational	rational	semi-decidable

Figure 10.3. This table lists to which language classes $\textsc{Reach}(L, T)$ and $\textsc{Reach}(L, \overline{T})$ belong.

Next, we consider the other aforementioned types of meta-transformations.

10.4.1 Recognizability

We start with considering meta-transformations which are closed under the behavioral equivalence $\equiv$. In other words, we study the effects of recognizable languages in the transformation monoid of a distributed queue.

From Corollary 4.8.11 we already know that for each action trace $\tau \in \mathbb{M}(\mathcal{E})$ there is a somewhat simple action trace $\sigma \in \overline{\mathbb{M}(\mathcal{A})}\mathbb{M}(\mathcal{A})\overline{\mathbb{M}(\mathcal{A})}$ behaving equivalently to τ. We can also compute such trace σ in polynomial time. We will use this fact to prove the following theorem:

Theorem 10.4.5. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $L \subseteq \mathbb{M}(\mathcal{A})$ be recognizable, and $T \subseteq \mathbb{M}(\mathcal{E})$ be recognizable and closed under behavioral equivalence $\equiv$, i.e., we have $T = \eta^{-1}(S)$ for a recognizable distributed queue language $S \subseteq \mathbb{T}(Q_{\mathcal{A}})$. Then $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$ are recognizable. In particular, we can construct asynchronous automata accepting $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$ from asynchronous automata accepting L and T in polynomial time.*

Proof idea. This proof is very similar to the proof of Theorem 10.2.5. Note that by Corollary 4.8.11 we have $\textsc{Reach}(L, T) = \textsc{Reach}(L, T \cap \overline{\mathbb{M}(\mathcal{A})}\mathbb{M}(\mathcal{A})\overline{\mathbb{M}(\mathcal{A})})$. Since the alphabets

A and $\overline{A}$ are disjoint sets, we know that $T \cap \overline{\mathbb{M}(\mathcal{A})}\mathbb{M}(\mathcal{A})\overline{\mathbb{M}(\mathcal{A})}$ is a union of polynomially many trace languages $\overline{K_1}K_2\overline{K_3}$ where $K_1, K_2, K_3 \subseteq \mathbb{M}(\mathcal{A})$ are (efficiently) recognizable. From the efficient closure properties (cf. Appendix A) we infer the efficient recognizability of the trace languages $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$. ◂

Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet. For a given recognizable trace language $T \subseteq \mathbb{M}(\mathcal{E})$ it is decidable whether T is closed under behavioral equivalence: to this end, we only have to check whether $\pi_i(T)$ is closed under the equations from Lemma 4.7.2 resp. Lemma 4.4.1 (if A_i is a singleton). Since $\mathbb{M}(\mathcal{E}{\restriction}_{A_i \cup \overline{A_i}})$ is isomorphic to the free monoid $(A_i \cup \overline{A_i})^*$, we can decide these properties as described in Theorem 5.4.8.

Theorem 10.4.5 similarly holds if we replace the given recognizable trace language $L \subseteq \mathbb{M}(\mathcal{A})$ of queue contents by a rational one:

Theorem 10.4.6. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $L \subseteq \mathbb{M}(\mathcal{A})$ be rational, and $T \subseteq \mathbb{M}(\mathcal{E})$ be recognizable and closed under behavioral equivalence $\equiv$. Then $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$ are rational. In particular, we can compute NFAs accepting $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$ from an NFA accepting L and an asynchronous automaton accepting T. If $\mathcal{A}$ is fixed, this construction is possible in polynomial time.*

Proof idea. Recall the proofs of Theorems 10.2.5 and 10.4.5. Then we see that the computation of $\textsc{Reach}(L, T)$ requires the efficient closure properties of recognizable trace languages (cf. Appendix A) as well as the following ones:

- the union of two rational trace languages is efficiently rational,
- the left- and right-quotient of a rational trace language wrt. a recognizable trace language is efficiently rational (for a fixed distributed alphabet $\mathcal{A}$), and
- the concatenation of two rational trace languages is efficiently rational.

Hence, $\textsc{Reach}(L, T)$ and $\textsc{BackReach}(L, T)$ are efficiently rational. ◂

10.4.2 Intermezzo: A Variation of Levi's Lemma

In the next subsection we want to consider meta-transformations looping through special languages of action sequences. In our proofs we will use Levi's lemma for traces multiple times. Concretely, we will utilize a special modification of this statement which is more suitable to our situation of an iterating distributed queue system.

So, let $\kappa, \lambda, \sigma_1, \sigma_2, \rho_1, \rho_2 \in \mathbb{M}(\mathcal{A})$ be traces satisfying $[\![\sigma_1\overline{\rho_1}\sigma_2\overline{\rho_2}]\!](\lambda) = \kappa \neq \bot$. According to Observation 3.4.7 we know that the trace equation $\lambda\sigma_1\sigma_2 = \rho_1\rho_2\kappa$ holds. Then Levi's lemma for traces (cf. Theorem 3.4.6) implies the existence of nine traces $\mu_{i,j} \in \mathbb{M}(\mathcal{A})$ with the following properties: (1) $\lambda = \mu_{1,1}\mu_{1,2}\mu_{1,3}$, (2) $\sigma_i = \mu_{i+1,1}\mu_{i+1,2}\mu_{i+1,3}$, (3) $\rho_j = \mu_{1,j}\mu_{2,j}\mu_{3,j}$, (4) $\kappa = \mu_{1,3}\mu_{2,3}\mu_{3,3}$, and (5) $\mu_{g,h} \parallel \mu_{i,j}$ if $g < i$ and $j < h$. Additionally, due to the definition of distributed queues, we cannot read letters from the queue before we have written them into the queue. Hence, there is no overlap of σ_2 and ρ_1 in the equation $\lambda\sigma_1\sigma_2 = \rho_1\rho_2\kappa$, i.e., we have $\mu_{3,1} = \varepsilon$. We can also extend this observation to an arbitrary alternation $\sigma_1\overline{\rho_1}\sigma_2\overline{\rho_2}\ldots\sigma_k\overline{\rho_k}$

of action traces. In this case we observe that the lower-left triangle of our visualization (cf. Figure 3.3) is empty. Concretely, we obtain the following result:

Corollary 10.4.7. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and let $\lambda, \kappa, \sigma_1, \ldots, \sigma_k, \rho_1, \ldots, \rho_k \in \mathbb{M}(\mathcal{A})$ be traces. We have $[\![\sigma_1\overline{\rho_1} \ldots \sigma_k\overline{\rho_k}]\!](\lambda) = \kappa \neq \bot$ if, and only if, there are traces $\mu_{i,j} \in \mathbb{M}(\mathcal{A})$ for $1 \leq i, j \leq k+1$ with the following properties:*

(1) $\lambda = \mu_{1,1}\mu_{1,2} \ldots \mu_{1,k+1}$,

(2) $\sigma_i = \mu_{i+1,1}\mu_{i+1,2} \ldots \mu_{i+1,k+1}$ *for each* $1 \leq i \leq k$,

(3) $\rho_j = \mu_{1,j}\mu_{2,j} \ldots \mu_{k+1,j}$ *for each* $1 \leq j \leq k$,

(4) $\kappa = \mu_{1,k+1}\mu_{2,k+1} \ldots \mu_{k+1,k+1}$,

(5) $\mu_{g,h} \parallel \mu_{i,j}$ *if* $g < i$ *and* $j < h$, *and*

(6) $\mu_{i,j} = \varepsilon$ *if* $i > j+1$.

	ρ_1	ρ_2	$\cdots$	ρ_{k-1}	ρ_k	κ
λ	$\mu_{1,1}$	$\mu_{1,2}$	$\cdots$	$\mu_{1,k-1}$	$\mu_{1,k}$	$\mu_{1,k+1}$
σ_1	$\mu_{2,1}$	$\mu_{2,2}$	$\cdots$	$\mu_{2,k-1}$	$\mu_{2,k}$	$\mu_{2,k+1}$
σ_2	ε	$\mu_{3,2}$	$\cdots$	$\mu_{3,k-1}$	$\mu_{3,k}$	$\mu_{3,k+1}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
σ_{k-1}	ε	ε	$\cdots$	$\mu_{k,k-1}$	$\mu_{k,k}$	$\mu_{k,k+1}$
σ_k	ε	ε	$\cdots$	ε	$\mu_{k+1,k}$	$\mu_{k+1,k+1}$

Figure 10.4. A visualization of the properties in Corollary 10.4.7. The properties in Theorem 3.4.6 hold. Additionally, the lower-left triangle (red) is empty.

Proof. First, assume $[\![\sigma_1\overline{\rho_1} \ldots \sigma_k\overline{\rho_k}]\!](\lambda) = \kappa \neq \bot$. Then there are $\kappa_0, \ldots, \kappa_k \in \mathbb{M}(\mathcal{A})$ with $[\![\sigma_1\overline{\rho_1} \ldots \sigma_j\overline{\rho_j}]\!](\lambda) = \kappa_j$ for each $0 \leq j \leq k$. In particular, we have $\kappa_0 = \lambda$ and $\kappa_k = \kappa$. We infer $[\![\sigma_j\overline{\rho_j}]\!](\kappa_{j-1}) = \kappa_j$ implying $\rho_j\kappa_j = \kappa_{j-1}\sigma_j$ for each $1 \leq j \leq k$ (by Observation 3.4.7). We construct the traces $\mu_{i,j} \in \mathbb{M}(\mathcal{A})$ by induction on j (i.e., column-wise in Figure 10.4). We do this with the help of iterated applications of Levi's Lemma for traces which yields traces $v_{i,1}^{(j)}, v_{i,2}^{(j)} \in \mathbb{M}(\mathcal{A})$ (for $1 \leq i \leq j+1$) where $\kappa_j = v_{1,2}^{(j)} \ldots v_{j+1,2}^{(j)}$ holds.

For $j = 0$ we set $v_{1,2}^{(0)} := \kappa_0 = \lambda$.

Now, let $1 \leq j \leq k$. Then we have $\kappa_{j-1} = v_{1,2}^{(j-1)} \ldots v_{j,2}^{(j-1)}$ implying

$$v_{1,2}^{(j-1)} \ldots v_{j,2}^{(j-1)}\, \sigma_j = \kappa_{j-1}\sigma_j = \rho_j\kappa_j\,.$$

By Levi's lemma (cf. Theorem 3.4.6) there are traces $v_{i,1}^{(j)}, v_{i,2}^{(j)} \in \mathbb{M}(\mathcal{A})$ (for $1 \leq i \leq j+1$) with the following properties:

(a) $v_{i,2}^{(j-1)} = v_{i,1}^{(j)} v_{i,2}^{(j)}$ for $1 \leq i \leq j+1$,

(b) $\sigma_j = v_{j+1,1}^{(j)} v_{j+1,2}^{(j)}$,

(c) $\rho_j = \nu_{1,1}^{(j)} \dots \nu_{j+1,1}^{(j)}$,

(d) $\kappa_j = \nu_{1,2}^{(j)} \dots \nu_{j+1,2}^{(j)}$, and

(e) $\nu_{g,1}^{(j)} \parallel \nu_{h,2}^{(j)}$ for each $g > h$.

We set $\mu_{i,j} := \nu_{i,1}^{(j)}$ for $1 \leq i \leq j+1$ and $\mu_{i,j} := \varepsilon$ for each $j+1 < i \leq k+1$. Finally, we set $\mu_{i,k+1} := \nu_{i,2}^{(k)}$.

By definition of the traces $\mu_{i,j}$ we already obtain the satisfaction of property (6). Now, we will prove the remaining properties:

(1) $\lambda = \nu_{i,2}^{(0)} = \nu_{1,1}^{(1)}\nu_{1,2}^{(1)} = \nu_{1,1}^{(1)}\nu_{1,1}^{(2)}\nu_{1,2}^{(2)} = \dots = \nu_{1,1}^{(1)} \dots \nu_{1,1}^{(k)}\nu_{1,2}^{(k)} = \mu_{1,1} \dots \mu_{1,k}\mu_{1,k+1}$

(2) $\sigma_i = \nu_{i+1,1}^{(i)}\nu_{i+1,2}^{(j)} = \dots = \nu_{i+1,1}^{(i)} \dots \nu_{i+1,1}^{(k)}\nu_{i+1,2}^{(k)} = \mu_{i+1,i} \dots \mu_{i+1,k+1} = \mu_{i+1,1} \dots \mu_{i+1,k+1}$

(3) $\rho_j = \nu_{1,1}^{(j)} \dots \nu_{j+1,1}^{(j)} = \mu_{1,j} \dots \mu_{j+1,j} = \mu_{1,j} \dots \mu_{k+1,j}$

(4) $\kappa = \kappa_k = \nu_{1,2}^{(k)} \dots \nu_{k+1,2}^{(k)} = \mu_{1,k+1} \dots \mu_{k+1,k+1}$

(5) If $g > h+1$ we have $\mu_{g,h} = \varepsilon$ which obviously implies our claim. Similarly, the claim is trivial if $i > j+1$. Hence, we assume $g \leq h+1$ and $i \leq j+1$. In this case we have $\mu_{g,h} = \nu_{g,1}^{(h)}$ (or $\mu_{g,h} = \nu_{g,2}^{(k)}$ if $h = k+1$) and $\mu_{i,j} = \nu_{i,1}^{(j)}$. We can prove that $\mu_{g,h}$ is an infix of $\nu_{g,2}^{(j)}$:

$$\nu_{g,2}^{(j)} = \nu_{g,1}^{(j+1)}\nu_{g,2}^{(j+1)} = \dots = \nu_{g,1}^{(j+1)} \dots \nu_{g,1}^{(h)}\nu_{g,2}^{(h)}.$$

Hence, we have either $\nu_{g,2}^{(j)} = \mu_{g,j+1} \dots \mu_{g,h}\nu_{g,2}^{(h)}$ (if $h \leq k$) or $\nu_{g,2}^{(j)} = \mu_{g,j+1} \dots \mu_{g,h}$ (if $h = k+1$). We infer $(\mathrm{Alph}(\mu_{g,h})\,\mathrm{M}) \subseteq (\mathrm{Alph}(\nu_{g,2}^{(j)})\,\mathrm{M})$ and, therefore,

$$(\mathrm{Alph}(\mu_{g,h})\,\mathrm{M}) \cap (\mathrm{Alph}(\mu_{i,j})\,\mathrm{M}) \subseteq (\mathrm{Alph}(\nu_{g,2}^{(j)})\,\mathrm{M}) \cap (\mathrm{Alph}(\nu_{i,1}^{(j)})) \overset{(e)}{=} \varnothing$$

by $i > g$. This finally implies $\mu_{g,h} \parallel \mu_{i,j}$.

Towards the converse implication, assume that the properties (1)-(6) hold. Then, by Levi's lemma for traces (cf. Theorem 3.4.6) we have $\lambda\sigma_1 \dots \sigma_k = \rho_1 \dots \rho_k\kappa$. We have to prove now that $[\![\sigma_1\overline{\rho_1} \dots \sigma_k\overline{\rho_k}]\!](\lambda) = \kappa$ holds (note that we cannot apply Observation 3.4.7 here, since it only states the converse implication). To this end, we prove by induction on $0 \leq i \leq k$ that

$$[\![\sigma_1\overline{\rho_1} \dots \sigma_i\overline{\rho_i}]\!](\lambda) = \prod_{g=1}^{i+1}(\mu_{g,i+1} \dots \mu_{g,k+1})$$

holds. So, let $i = 0$. Then we have $[\![\varepsilon]\!](\lambda) = \lambda = \mu_{1,1} \dots \mu_{1,k+1}$ by (1). Next, let $1 \leq i \leq k$. Then we obtain

$$
\begin{aligned}
&[\![\sigma_1\overline{\rho_1}\dots\sigma_i\overline{\rho_i}]\!](\lambda)\\
&= [\![\sigma_i\overline{\rho_i}]\!]([\![\sigma_1\overline{\rho_1}\dots\sigma_{i-1}\overline{\rho_{i-1}}]\!](\lambda))\\
&= [\![\mu_{i+1,1}\dots\mu_{i+1,k+1}\overline{\mu_{1,i}\dots\mu_{k+1,i}}]\!]\left(\prod_{g=1}^{i}(\mu_{g,i}\dots\mu_{g,k+1})\right) && \text{(by (2), (3), and i.h.)}\\
&= [\![\mu_{i+1,i}\dots\mu_{i+1,k+1}\overline{\mu_{1,i}\dots\mu_{k+1,i}}]\!]\left(\prod_{g=1}^{i}(\mu_{g,i}\dots\mu_{g,k+1})\right) && \text{(by (6))}\\
&= [\![\overline{\mu_{1,i}\dots\mu_{k+1,i}}]\!]\left(\prod_{g=1}^{i+1}(\mu_{g,i}\dots\mu_{g,k+1})\right)\\
&= \prod_{g=1}^{i+1}(\mu_{g,i+1}\dots\mu_{g,k+1})\,. && \text{(by (5))}
\end{aligned}
$$

Hence, we have $[\![\sigma_1\overline{\rho_1}\dots\sigma_k\overline{\rho_k}]\!](\lambda) = \prod_{g=1}^{k+1}\mu_{g,k+1} = \kappa$ by (4). ◂

10.4.3 Alternations

In this subsection we want to consider meta-transformations alternating between two given languages of action traces. To this end, consider a distributed system consisting of three processes 1, 2, and 3. The process 1 can handle requests of type a and b, process 2 handles a and c, and process 3 handles requests of type c. In other words, the underlying distributed alphabet $\mathcal{A}$ has the dependence graph $b - a - c$. Now, assume that users can send several sequences of requests. For example, sequences ab, ac, and c are send to the system in an arbitrary order. The processes execute several of their requests one after another - in order of incoming. Then we may ask for the set of reachable configurations of our system. In this concrete situation we have to compute $\textsc{Reach}(\{\varepsilon\}, (\{ab, ac, c\}\{\overline{a}, \overline{b}, \overline{c}\}^*)^*)$. Using the approach from Boigelot et al., we will never obtain the result $\textsc{Reach}(\{\varepsilon\}, (\{ab, ac, c\}\{\overline{a}, \overline{b}, \overline{c}\}^*)^*) = \{[b], \varepsilon\} \cdot \{[ab], [ac], [c]\}^*$, since we consider multiple interlaced loops here.

Hence, the aim of this section is to generalize our result from Theorem 10.2.12 stating that $\textsc{Reach}(L, (W\overline{R})^*)$ (in single-queue systems) is efficiently regular whenever $L, W, R \subseteq A^*$ are regular languages. Here, we consider in which cases $\textsc{Reach}(L, (W\overline{R})^*)$ effectively preserves the recognizability of the queue contents $L \subseteq \mathbb{M}(\mathcal{A})$ for arbitrary distributed alphabets $\mathcal{A} = (A, P, \mathrm{M})$ (note that the aforementioned result was stated for distributed alphabets having a complete dependence graph).

In the general case recognizability of the trace languages $L, W, R \subseteq \mathbb{M}(\mathcal{A})$ does not necessarily imply recognizability of $\textsc{Reach}(L, (W\overline{R})^*)'$. This is due to the fact that the class of recognizable trace languages is not closed under iteration. For example, let $a, b \in A$ with $a \parallel b$, $L = R := \{\varepsilon\}$, and $W := \{[ab]\}$. Then we obtain $\textsc{Reach}(L, (W\overline{R})^*) = W^* = [ab]^*$ which is not recognizable in $\mathbb{M}(\mathcal{A})$ since $[ab]$ is not connected. However, if the recognizable trace language W to be iterated also is connected then W^* is efficiently recognizable as well (at least for a fixed distributed alphabet $\mathcal{A}$, cf. Appendix A). Hence, in our generalization of Theorem 10.2.12 we also require this restriction to $W \subseteq \mathbb{M}(\mathcal{A})$. Note that we do not require connectivity for the trace languages L and R.

Concretely, we will prove the following statement:

Theorem 10.4.8. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, and $L, W, R \subseteq \mathbb{M}(\mathcal{A})$ be recognizable such that W is connected. Then $\textsc{Reach}(L, (W\overline{R})^*)$ is recognizable. In particular, we can construct an asynchronous automaton accepting $\textsc{Reach}(L, (W\overline{R})^*)$ from asynchronous automata accepting L, W, and R. If $\mathcal{A}$ is fixed, this construction is possible in polynomial time.*

Similar to the single-queue case, the proof is based on the following ideas:

(1) Instead of deciding which trace $\sigma \in W$ we write into the queues per round, we just keep one token. To this end, we prepend each trace $\sigma \in W$ with a new letter to mark its beginning. In contrast to the single-queue case, we need (for technical reasons) more than just one such begin marker, but each $\sigma \in W$ will be prepended by just one of them. However, independent of the concrete begin marker we always keep the same type of token. When reading letters, we remove such tokens and decide at this moment, which trace we have written into our distributed queue.

(2) Since we only count tokens representing the number of traces from $\sigma \in W$ to be contained in our distributed queue, we can simulate this with the help of a counter, a unary queue, or a unary pushdown.

Hence, our distributed queue system can be simulated by a pushdown automaton $\mathfrak{P}$. Since the map $\mathrm{post}^*_{\mathfrak{P}} \colon \mathrm{Conf}_{\mathfrak{P}} \to \mathrm{Conf}_{\mathfrak{P}}$ efficiently preserves regularity (by Theorem 9.2.1), we will finally obtain the efficient recognizability of $\textsc{Reach}(L, (W\overline{R})^*)$.

We start our proof by describing the construction of the pushdown automaton. Afterwards, we prove the correctness of this theorem on Page 216.

The Reduction to Pushdown Automata

Towards the construction of the pushdown automaton simulating our alternating distributed queue system, we should first recall the abstraction of the single-queue system's configurations. Concretely, we have abstracted the content w of a single queue with the help of two states marking the begin and end of a run labeled with w in an NFA accepting LW^* (note that all reachable contents are infixes of LW^* in this case). We stored this information in the control states of the constructed PDA. Moreover, in the PDA's stack we counted the number of words from W to be contained in the queue's content. We simplified this counting process by introducing a special begin marker $\$ \notin A$ which we prepended to each word from W. Since the constructed PDA uses only one stack symbol, it is essentially a counter automaton without zero-tests.

Here, we want to construct a pushdown automaton with similar semantics of its configurations. First, we want to introduce begin markers of the traces in W. Unfortunately, the situation is way more complicated than for single queues. This is due to the partial commutations induced by the distributed alphabet $\mathcal{A}$. For example, consider a distributed alphabet $\mathcal{A}$ with dependence graph $a - b - c$, $L = \{[\varepsilon]\}$, $W = \{[a], [b], [c]\}$, and $R = \mathbb{M}(\mathcal{A})$. A possible computation of a distributed queue is the following one:

$$[\varepsilon] \xrightarrow{a}_{\mathfrak{Q}} [a] \xrightarrow{\overline{\varepsilon}}_{\mathfrak{Q}} [a] \xrightarrow{c}_{\mathfrak{Q}} [ac] \xrightarrow{\overline{c}}_{\mathfrak{Q}} [a] \xrightarrow{b}_{\mathfrak{Q}} [ab] \xrightarrow{\overline{a}}_{\mathfrak{Q}} [b] .$$

In other words, even if we have first written a and then c we are able to first read c and then a. So, a possible begin marker of a has to be independent of the one of c. However, the begin markers of a and b (or c and b) have to be dependent since we cannot read b from a distributed queue with content $[ab]$.

We will see later in the correctness proof that a good set of begin markers consists of letters $\$_{C,i}$ where $C \subseteq A$ is a connected set of letters and $i \in P$. Concretely, we will prepend a trace $\sigma \in W$ with the letter $\$_{C,i}$ if C is exactly the (connected) set of letters occurring in σ and if σ start with a letter in A_i. So, we first construct a modified distributed alphabet $\mathcal{A}' = (A', P', \mathrm{M}')$ with

- $A' := A \cup B$ where $B := \{\$_{C,i} \mid C \subseteq A \text{ connected}, i \in C\,\mathrm{M}\} \cup \{\$_{\varnothing,0}\}$ is disjoint from A,
- $P' := \{i, \hat{i} \mid i \in P\} \cup \{0\}$ where $\hat{P} := \{\hat{i} \mid i \in P\}$ is a disjoint copy of P and $0 \notin P \cup \hat{P}$, and
- $\mathrm{M}' := \mathrm{M} \cup \{(\$_{C,i}, i), (\$_{C,i}, \hat{j}) \mid \$_{C,i} \in B, j \in C\,\mathrm{M}\}$ (i.e., we add $\$_{C,i}$ to process i and to the copy $\hat{j}$ of any $j \in C\,\mathrm{M}$).

This distributed alphabet $\mathcal{A}'$ also induces a distributed alphabet $\mathcal{E}' := (\Sigma', P', \mathrm{M}')$ extended by read actions where we have $\Sigma' = \Sigma \cup B \cup \overline{B} = A' \cup \overline{A'}$. We can understand $\$_{C,i} \in B$ as follows: in C we store the letters occurring in a trace σ which starts with this begin marker $\$_{C,i}$. Additionally, the pushdown automaton $\mathfrak{P}$ (to be constructed) guesses a process $i \in P$. When it starts reading σ from the distributed queue's content it first reads $\$_{C,i}$ and afterwards a letter from process i.

Accordingly, we next introduce trace languages $W_{C,i}$ where $\varnothing \subsetneq C \subseteq A$ is connected and $i \in C\,\mathrm{M}$. Concretely, $W_{C,i}$ contains all traces $\sigma \in W$ with $\mathrm{Alph}(\sigma) = C$ (i.e., σ contains exactly the letters from C) and $\sigma \in \zeta_{\mathcal{A}'}(A_i) \cdot \mathbb{M}(\mathcal{A})$ (i.e., σ starts with a letter in A_i) which we prepend with the letter $\$_{C,i}$. Formally, we set

$$W_{C,i} := \$_{C,i} \cdot \left(W \cap \zeta_{\mathcal{A}'}(A_i) \cdot \mathbb{M}(\mathcal{A}) \cap \bigcap_{a \in C} \mathbb{M}(\mathcal{A}{\restriction_C}) \cdot a \cdot \mathbb{M}(\mathcal{A}{\restriction_C}) \right).$$

Note that due to the closure properties of the class of recognizable trace languages, we can compute an asynchronous automaton accepting $W_{C,i}$ in polynomial time from an asynchronous automaton accepting W, the connected set $C \subseteq A$, and $i \in C\,\mathrm{M}$. Additionally, by the definition of our modified distributed alphabet, $W_{C,i}$ is still connected. We also set $W_{\varnothing,0} := \{\$_{\varnothing,0}\}$ if $\varepsilon \in W$ and $W_{\varnothing,0} := \varnothing$ otherwise. In other words, we also prepend a special begin marker to the empty trace (iff it exists in W). Finally, we set $W' := \bigcup_{\$_{C,i} \in B} W_{C,i}$. This language is still recognizable in $\mathbb{M}(\mathcal{A}')$ and connected. We can compute this trace language in time polynomial in the size of an asynchronous automaton accepting W (and in time exponential in the size of $\mathcal{A}$).

Example 10.4.9. Let $\mathcal{A} = (\{a, b\}, \{1, 2\}, \{(a, 1), (b, 2)\})$ be a distributed alphabet and $W = \{[aa], [b]\}$. Then we have $B = \{\$_{\{a\},1}, \$_{\{b\},2}, \$_{\varnothing,0}\}$, $W_{\varnothing,0} = \varnothing$, $W_{\{a\},1} = \{[\$_{\{a\},1}aa]\}$, and $W_{\{b\},2} = \{[\$_{\{b\},2}b]\}$. ⌟

Additionally, we set $R' := \pi_A^{-1}(R)$, i.e., we insert letters from B into the traces from R at arbitrary positions. Since the class of recognizable languages is closed under inverse homomorphisms, the trace language R' also is efficiently recognizable. By the choice of $\mathcal{A}'$, W', and R' we can claim the following equation:

$$\textsc{Reach}(L, (W\overline{R})^*) = \pi_A(\textsc{Reach}(L, (W'\overline{R'})^*)).$$

In other words, the insertion of the letters from B into W and R does not change the behavior of our distributed queue system. We will see the correctness of this equation later in this proof.

Now, we want to deduce an abstraction of the contents of our distributed queue. To this end, we consider a non-failing computation $\tau \in (W'\overline{R'})^*$ of the modified distributed queue system with initial content $\lambda \in L$, i.e., $[\![\tau]\!](\lambda) \neq \bot$. Let $\kappa := [\![\tau]\!](\lambda)$. Then by Observation 3.4.7 we obtain $\lambda \cdot \mathrm{wrt}(\tau) = \overline{\mathrm{rd}(\tau)} \cdot \kappa$ where $\mathrm{wrt}(\tau) \in W'^*$ and $\mathrm{rd}(\tau) \in R'^*$ holds. Hence, κ is an infix of the trace $\lambda \cdot \mathrm{wrt}(\tau) \in LW'^*$ (note that this also holds for any intermediate result on the computation of $[\![\tau]\!](\lambda)$). Since LW'^* is efficiently recognizable, there is an asynchronous automaton $\mathfrak{C} = (\vec{Q}_{\mathfrak{C}}, A', I_{\mathfrak{C}}, \Delta_{\mathfrak{C}}, F_{\mathfrak{C}})$ accepting LW'^*. W.l.o.g. we can assume that each initial state $\vec{\imath} \in I_{\mathfrak{C}}$ reaches a final state $\vec{f} \in F_{\mathfrak{C}}$ and each final state can be reached from an initial state[xxi]. Since $\lambda \cdot \mathrm{wrt}(\tau) \in T(\mathfrak{C})$ there is an asynchronous run $\Pi = (\vec{\imath} \xrightarrow{\lambda \cdot \mathrm{wrt}(\tau)}_{\mathfrak{C}} \vec{f})$ in $\mathfrak{C}$ labeled with $\lambda \cdot \mathrm{wrt}(\tau)$ from an initial state $\vec{\imath} \in I_{\mathfrak{C}}$ to an accepting state $\vec{f} \in F_{\mathfrak{C}}$.

Recall that $\tau \in (W'\overline{R'})^*$ holds. Since this trace language is efficiently rational, there is an NFA $\mathfrak{T} = (Q_{\mathfrak{T}}, \Sigma', I_{\mathfrak{T}}, \Delta_{\mathfrak{T}}, F_{\mathfrak{T}})$ accepting $(W'\overline{R'})^*$. For technical reasons we assume that $\mathfrak{T}$ is constructed from asynchronous automata accepting W' and $\overline{R'}$, resp., using the classical constructions for concatenation and iteration of regular languages (cf. Figure 10.5).

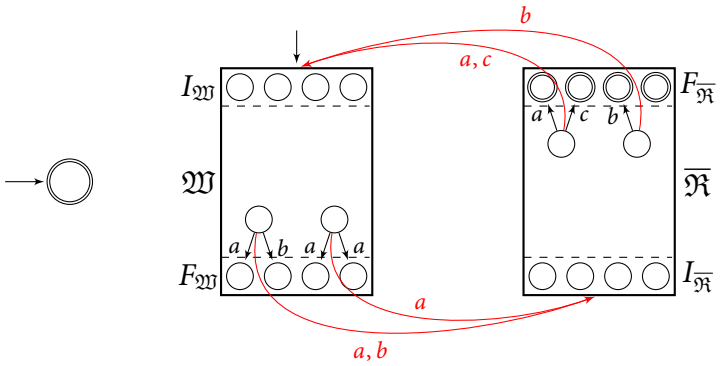

Figure 10.5. A visualization of the construction of the NFA $\mathfrak{T}$ from asynchronous automata $\mathfrak{W}$ and $\overline{\mathfrak{R}}$ accepting W' and $\overline{R'}$, resp. Note that the final states of $\mathfrak{W}$ are not accepting and the initial states of $\overline{\mathfrak{R}}$ are not initial anymore.

Due to our construction we can observe the following properties of $\mathfrak{T}$:

(a) For $[w_1] \in W'$ and $s_1, s_2 \in Q_{\mathfrak{T}}$ with $s_1 \xrightarrow{w_1}_{\mathfrak{T}} s_2$ we also have $s_1 \xrightarrow{w_2}_{\mathfrak{T}} s_2$ for each $w_2 \approx_{\mathcal{A}'} w_1$.

(b) For $[r_1] \in R'$ and $s_1, s_2 \in Q_{\mathfrak{T}}$ with $s_1 \xrightarrow{\overline{r_1}}_{\mathfrak{T}} s_2$ we also have $s_1 \xrightarrow{\overline{r_2}}_{\mathfrak{T}} s_2$ for each $r_2 \approx_{\mathcal{A}'} r_1$.

(c) For $w_1, \ldots w_k, r_1, \ldots, r_k \in A'^*$ with $I_{\mathfrak{T}} \xrightarrow{w_1\overline{r_1}\ldots w_k\overline{r_k}}_{\mathfrak{T}} F_{\mathfrak{T}}$ we have $[w_1], \ldots, [w_k] \in W'$ and $[r_1], \ldots, [r_k] \in R'$.

In other words, the sequences of write actions and read actions, resp., are closed under partial commutations $\approx_{\mathcal{A}}$. Additionally, the action sequences from W' and $\overline{R'}$ are strictly separated in $\mathfrak{T}$.

[xxi] Recall that the NFA $\mathfrak{C}$ in the proof of Theorem 10.4.8 was trim in the sense that each state is reachable from an initial state and reaches a final state. Now, we only require that the initial and accepting states have this property.

Since $\tau \in T(\mathfrak{T})$ there are letters $\alpha_1, \dots, \alpha_m \in \Sigma'$ with $[\alpha_1 \dots \alpha_m] = \tau$ and and accepting run $\iota \xrightarrow{\alpha_1 \dots \alpha_m}_{\mathfrak{T}} f$ of τ in $\mathfrak{T}$ from an initial state $\iota \in I_{\mathfrak{T}}$ to an accepting state $f \in F_{\mathfrak{T}}$. Let $\iota = s_0, s_1, \dots, s_m = f$ be the intermediate states of this run. Additionally, since $\alpha_1 \dots \alpha_m = [\tau]$ there are intermediate queue contents $\lambda_0, \lambda_1, \dots, \lambda_m \in \mathbb{M}(\mathcal{A}')$ with $\lambda_0 = \lambda$, $\lambda_{j+1} = [\![\alpha_{j+1}]\!](\lambda_j)$ for each $0 \leq j < m$ (implying $\lambda_j \mathrm{wrt}(\alpha_{j+1}) = \mathrm{rd}(\alpha_{j+1})\lambda_{j+1}$), and $\lambda_m = \kappa$. Then we want to abstract the intermediate configuration (s_j, λ_j) of our distributed queue system (for $0 \leq j \leq m$) with the following information:

(1) the control state s_j of our automaton,

(2) two intermediate states Π_{μ_j} and Π_{ν_j} marking the begin and end of a subrun labeled with λ_j of our asynchronous run Π, and

(3) the number $c_j := |\lambda_j|_B$ of traces from W' to be contained in λ_j.

Initially, we abstract the configuration (s_0, λ_0) by $(s_0, \Pi_\varepsilon, \Pi_\lambda, 0)$ since there is a run in $\mathfrak{C}$ labeled with λ from Π_ε to Π_λ and since $|\lambda_0|_B = |\lambda|_B = 0$ by $\lambda \in L \subseteq \mathbb{M}(\mathcal{A})$. Next, we construct the abstraction of (s_{j+1}, λ_{j+1}) from (s_j, λ_j) as follows: let $(s_j, \Pi_{\mu_j}, \Pi_{\nu_j}, c_j)$ be the abstraction of (s_j, λ_j). By the choice of our run in $\mathfrak{T}$ we have an edge $s_j \xrightarrow{\alpha_{j+1}}_{\mathfrak{T}} s_{j+1}$. Additionally, we distinguish the following cases:

(1) If $\alpha_{j+1} = a \in A'$, there is an edge $\Pi_{\nu_j} \xrightarrow{a}_{\mathfrak{C}} \Pi_{\nu_j a}$. Since the asynchronous run from Π_{μ_j} to Π_{ν_j} is labeled by λ_j, we can extend this run with this a-edge to a run from Π_{μ_j} to $\Pi_{\nu_j a}$ labeled by $\lambda_j a = \lambda_{j+1}$. Additionally, if $a \in B$ holds, the number of B-letters in λ_j increases. Hence, we can abstract (s_{j+1}, λ_{j+1}) by $(s_{j+1}, \Pi_{\mu_j}, \Pi_{\nu_j a}, c_j + |a|_B)$.

(2) If $\alpha_{j+1} = \overline{a} \in \overline{A'}$, the asynchronous run from Π_{μ_j} to Π_{ν_j} starts with the a-edge $\Pi_{\mu_j} \xrightarrow{a}_{\mathfrak{C}} \Pi_{\mu_j a}$ since $\lambda_j = a\lambda_{j+1}$ (note that our computation does not end up in the error state $\bot$). Hence, there is a run from $\Pi_{\mu_j a}$ to Π_{ν_j} labeled with λ_{j+1}. If $a \in B$ holds, the number of B-letters in λ_j decreases. Therefore, the appropriate abstraction of (s_{j+1}, λ_{j+1}) is $(s_{j+1}, \Pi_{\mu_j a}, \Pi_{\nu_j}, c_j - |a|_B)$ in this case.

All in all, $(s_j, \Pi_{\mu_j}, \Pi_{\nu_j}, c_j)$ is a suitable abstraction of the distributed queue system's configuration (s_j, λ_j). These information can be simulated with the help of a pushdown automaton $\mathfrak{P}$. In this case, the control states of $\mathfrak{P}$ are composed of the states s_j, Π_{μ_j}, and Π_{ν_j} while the stack contains c_j many letters (for a better differentiation the stack's content is $\#^{c_j}$ in this case). Note that this PDA is essentially a (partially blind) one-counter automaton.

We define the pushdown automaton $\mathfrak{P} := (Q_{\mathfrak{P}}, \Sigma', \mathcal{P}_{\{\#\}}, I_{\mathfrak{P}}, \varepsilon, \Delta_{\mathfrak{P}}, F_{\mathfrak{P}})$ as follows:

- $Q_{\mathfrak{P}} := Q_{\mathfrak{T}} \times \vec{\boldsymbol{Q}}_{\mathfrak{C}} \times \vec{\boldsymbol{Q}}_{\mathfrak{C}}$,
- $I_{\mathfrak{P}} := I_{\mathfrak{T}} \times I_{\mathfrak{C}} \times Q_L$ where $Q_L := \{\vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}_{\mathfrak{C}} \mid \exists \lambda \in L \colon I_{\mathfrak{C}} \xrightarrow{\lambda}_{\mathfrak{C}} \vec{\boldsymbol{q}}\}$,
- $F_{\mathfrak{P}} := F_{\mathfrak{T}} \times \vec{\boldsymbol{Q}}_{\mathfrak{C}} \times F_{\mathfrak{C}}$, and
- $\Delta_{\mathfrak{P}}$: we first introduce the homomorphism $\vartheta \colon \mathbb{M}(\mathcal{A}') \to \#^*$ induced by $\vartheta(a) = \varepsilon$ if $a \in A$ and $\vartheta(a) = \#$ if $a \in B$ holds (in other words, ϑ projects A' to B and renames the letters from B to # afterwards). Then $\Delta_{\mathfrak{P}}$ consists of the following transitions:

  (W) *Simulate writing of the letter $a \in A'$ into the distributed queue:* $((s, \vec{\boldsymbol{p}}, \vec{\boldsymbol{q}}), a, \vartheta(a), (s', \vec{\boldsymbol{p}}, \vec{\boldsymbol{q}}')) \in \Delta_{\mathfrak{P}}$ if, and only if, $(s, a, s') \in \Delta_{\mathfrak{T}}$ and $(\vec{\boldsymbol{q}}, a, \vec{\boldsymbol{q}}') \in \Delta_{\mathfrak{C}}$

(R) *Simulate reading of the letter $a \in A'$ from the distributed queue:* $((s, \vec{\boldsymbol{p}}, \vec{\boldsymbol{q}}), \overline{a}, \overline{\vartheta(a)}, (s', \vec{\boldsymbol{p}}', \vec{\boldsymbol{q}})) \in \Delta_{\mathfrak{P}}$ if, and only if, $(s, \overline{a}, s') \in \Delta_{\mathfrak{C}}$, $(\vec{\boldsymbol{p}}, a, \vec{\boldsymbol{p}}') \in \Delta_{\mathfrak{C}}$, and $T(\mathfrak{C}_{\vec{\boldsymbol{p}}' \to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A}) \neq \varnothing$.

Note that the condition "$T(\mathfrak{C}_{\vec{\boldsymbol{p}}' \to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A}) \neq \varnothing$" in transition (R) is required in the correctness proof of our construction. It ensures that the first letter the simulated distributed queue system reads from any trace $\sigma \in W'$ is its begin marker $\$_{C,i} \in B$.

The Semantics of the Pushdown Automaton

Now, we assign to the configuration $((s, \vec{\boldsymbol{p}}, \vec{\boldsymbol{q}}), \#^c) \in \mathrm{Conf}_{\mathfrak{P}}$ the set of all traces having the abstraction $(s, \vec{\boldsymbol{p}}, \vec{\boldsymbol{q}}, c)$. These are all traces being the labeling of an asynchronous run from $\vec{\boldsymbol{p}}$ to $\vec{\boldsymbol{q}}$ in $\mathfrak{C}$ and containing c appearances of letters from B (recall that these letters mark the beginning of any trace from W'). Formally, our assignment is the mapping $[\![.]\!]_{\mathfrak{P}} \colon \mathrm{Conf}_{\mathfrak{P}} \to 2^{\mathbb{M}(\mathcal{A}')}$ with

$$[\![((s, \vec{\boldsymbol{p}}, \vec{\boldsymbol{q}}), \#^c)]\!]_{\mathfrak{P}} := T(\mathfrak{C}_{\vec{\boldsymbol{p}} \to \vec{\boldsymbol{q}}}) \cap \vartheta^{-1}(\#^c)$$

for each $s \in Q_{\mathfrak{T}}$, $\vec{\boldsymbol{p}}, \vec{\boldsymbol{q}} \in \vec{\boldsymbol{Q}}_{\mathfrak{C}}$, and $c \in \mathbb{N}$ where ϑ is the homomorphism mapping letters from A to ε and letters from B to $\#$.

Our next aim is to prove that the semantics of all reachable accepting configurations of our constructed pushdown automaton $\mathfrak{P}$ agrees with the set $\textsc{Reach}(L, (W'\overline{R'})^*)$. Concretely, we prove the equality of the projections to the primary alphabet A of these two trace languages. This is the following statement:

Proposition 10.4.10. $\textsc{Reach}(L, (W\overline{R})^*) = \bigcup_{\gamma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}} \pi_A([\![\gamma]\!]_{\mathfrak{P}})$.

Later this proposition will turn out to be a key component of the correctness proof of Theorem 10.4.8. We prove this proposition with the help of two lemmas each stating one inclusion.

We start with proving that any reachable queue content is associated to the semantics of a reachable accepting configuration of $\mathfrak{P}$. The proof of this inclusion proceeds very similar to the proof of Lemma 10.2.15 but is a bit more involved. So, let $\tau \in (W\overline{R})^*$ and $\lambda \in L$ with $[\![\tau]\!](\lambda) \neq \bot$. First, we have to insert begin markers $\$_{C,i} \in B$ and their corresponding read actions $\overline{\$_{C,i}} \in \overline{B}$ at the appropriate positions in τ such that the resulting trace τ' belongs to $(W'\overline{R'})^*$ and $\pi_A([\![\tau']\!](\lambda)) = [\![\tau]\!](\lambda) \neq \bot$ holds. Afterwards we construct an accepting run of the PDA $\mathfrak{P}$ such that the j^{th} intermediate result on computation of $[\![\tau']\!](\lambda)$ belongs to the semantics of the j^{th} configuration of the constructed run of our PDA. Note that especially the consideration of read actions is way more complicated than in the single-queue case, since we also have to show the property "$T(\mathfrak{C}_{\vec{\boldsymbol{p}}' \to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A}) \neq \varnothing$".

Lemma 10.4.11. *Let $\tau \in (W\overline{R})^*$ and $\lambda \in L$ with $[\![\tau]\!](\lambda) \neq \bot$. Then there is a configuration $\gamma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$ with $[\![\tau]\!](\lambda) \in \pi_A([\![\gamma]\!]_{\mathfrak{P}})$.*

Proof. Let $\tau = \sigma_1\overline{\rho_1}\sigma_2\overline{\rho_2}\dots\sigma_k\overline{\rho_k}$ with $\sigma_1, \sigma_2, \dots, \sigma_k \in W$ and $\rho_1, \rho_2, \dots, \rho_k \in R$. Additionally, let $\kappa := [\![\tau]\!](\lambda) \neq \bot$. By Corollary 10.4.7 we can factorize $\lambda, \sigma_1, \dots, \sigma_k, \rho_1, \dots, \rho_k, \kappa \in \mathbb{M}(\mathcal{A})$ into traces $[w_{i,j}] \in \mathbb{M}(\mathcal{A})$ with the properties (1)-(6). Now, we insert $\$_{C,x} \in B$ into the $w_{i,j}$ resulting in the following words $w'_{i,j} \in \Sigma'^*$:

$$w'_{i,j} := \begin{cases} \$_{C,x} w_{i,j} & \text{if } i > 1, w_{i,1} \dots w_{i,j-1} = \varepsilon, C = \mathrm{Alph}(w_{i,j} \dots w_{i,k+1}), x \in C\,\mathrm{M}, w_{i,j} \in A_x A^* \\ \$_{\varnothing,0} & \text{if } i > 1, j = k+1, w_{i,1} \dots w_{i,k+1} = \varepsilon \\ w_{i,j} & \text{otherwise}. \end{cases}$$

	ρ_1	ρ_2	ρ_3	$\cdots$	ρ_{k-1}	ρ_k	κ
λ	$w_{1,1}$	$w_{1,2}$	$w_{1,3}$	$\cdots$	$w_{1,k-1}$	$w_{1,k}$	$w_{1,k+1}$
σ_1	$\$_{C_1,i_1} w_{2,1}$	$w_{2,2}$	$w_{2,3}$	$\cdots$	$w_{2,k-1}$	$w_{2,k}$	$w_{2,k+1}$
σ_2	ε	ε	ε	$\cdots$	ε	ε	$\$_{\varnothing,0}\varepsilon$
σ_3	ε	ε	$\$_{C_3,i_3} w_{4,3}$	$\cdots$	$w_{4,k-1}$	$w_{4,k}$	$w_{4,k+1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
σ_{k-1}	ε	ε	ε	$\cdots$	ε	ε	$\$_{C_{k-1},i_{k-1}} w_{k,k+1}$
σ_k	ε	ε	ε	$\cdots$	ε	$\$_{C_k,i_k} w_{k+1,k}$	$w_{k+1,k+1}$

Figure 10.6. Visualization of the construction of the $w'_{i,j}$'s, the σ'_i's, the ρ'_j's, and κ'.

In other words, as depicted in Figure 10.6 we prepend a letter $\$_{C,x} \in B$ to each σ_i (i.e., we prepend this letter to each row except for the first one). Then we define $\sigma'_i := [w'_{i+1,1} \dots w'_{i+1,k+1}]$ for each $1 \leq i \leq k$, $\rho'_j := [w'_{1,j} \dots w'_{k+1,j}]$ for each $1 \leq j \leq k$, and $\kappa' := [w'_{1,k+1} \dots w'_{k+1,k+1}]$. We obtain the following properties:

- $\sigma'_i = [w'_{i+1,1} \dots w'_{i+1,k+1}] = [\$_{C,x} w_{i+1,1} \dots w_{i+1,k+1}] \in W_{C,x} \subseteq W'$ for appropriate $C \subseteq A$ and $x \in P \cup \{0\}$.
- $\pi_A(\rho'_j) = \pi_A([w'_{1,j} \dots w'_{k+1,j}]) = [w_{1,j} \dots w_{k+1,j}] \in R$ implying $\rho'_j \in \pi_A^{-1}(R) = R'$.
- $\lambda = [w_{1,1} \dots w_{1,k+1}] = [w'_{1,1} \dots w'_{1,k+1}]$ (note that $w_{i,j} \neq w'_{i,j}$ requires $i > 1$).

Then the traces $[w'_{i,j}]$ still satisfy the properties from Corollary 10.4.7 implying the equation $\kappa' = [\![\sigma'_1\overline{\rho'_1}\dots\sigma'_k\overline{\rho'_k}]\!](\lambda) \neq \bot$. Now, we will prove that $\kappa' \in [\![\gamma]\!]_{\mathfrak{P}}$ holds for a configuration $\gamma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$. This will finally imply $\kappa = \pi_A(\kappa') \in \pi_A([\![\gamma]\!]_{\mathfrak{P}})$.

Since $\lambda\sigma'_1 \dots \sigma'_k \in LW'^* = T(\mathfrak{C})$ there is an asynchronous run $\Pi = (\vec{\boldsymbol{\imath}} \xrightarrow{\lambda\sigma'_1\dots\sigma'_k}_{\mathfrak{C}} \vec{\boldsymbol{f}})$ in $\mathfrak{C}$ from an initial state $\Pi_\varepsilon = \vec{\boldsymbol{\imath}} \in I_{\mathfrak{C}}$ to a final state $\Pi_{\lambda\sigma'_1\dots\sigma'_k} = \vec{\boldsymbol{f}} \in F_{\mathfrak{C}}$ with labeling $\lambda\sigma'_1 \dots \sigma'_k$. Additionally, since $\sigma'_1\overline{\rho'_1}\dots\sigma'_k\overline{\rho'_k} \in (W'\overline{R'})^* = T(\mathfrak{T})$ holds, there are letters $\alpha_1, \dots, \alpha_n \in \Sigma'$ with $[\alpha_1 \dots \alpha_n] = \sigma'_1\overline{\rho'_1}\dots\sigma'_k\overline{\rho'_k}$ and a run $s_0, \dots, s_n$ in $\mathfrak{T}$ from an initial state $s_0 \in I_{\mathfrak{T}}$ to an accepting state $s_n \in F_{\mathfrak{T}}$ with $s_{j-1} \xrightarrow{\alpha_j}_{\mathfrak{T}} s_j$ for each $1 \leq j \leq n$. We can assume that

$$\mathrm{rd}(\alpha_1 \dots \alpha_n) = w'_{1,1} \dots w'_{k+1,1} \dots w'_{1,k} \dots w'_{k+1,k}$$

holds in the free monoid A'^*[xxii], i.e., we read the letters column-wise in Figure 10.6 (except for the last column). This ensures that the first letter we read from any σ'_j (where $1 \leq j \leq k$) is its begin marker $\$_{C_j,i_j} \in B$.

[xxii] Note that we are able to choose the letters $\alpha_1, \dots, \alpha_n \in \Sigma'$ and the run $s_0, \dots, s_n$ such that this equation holds. This is possible due to property (b) of the NFA $\mathfrak{T}$.

From $\bot \neq \kappa' = [\![\sigma_1'\overline{\rho_1'}\dots\sigma_k'\overline{\rho_k'}]\!](\lambda) = [\![\alpha_0\dots\alpha_n]\!](\lambda)$ we also obtain traces $\lambda_0,\dots,\lambda_n \in \mathbb{M}(\mathcal{A}')$ with $\lambda_0 = \lambda$, $[\![\alpha_{j+1}]\!](\lambda_j) = \lambda_{j+1} \neq \bot$ for each $0 \leq j < n$, and $\lambda_n = \kappa'$.

Now, we want to construct a run $\gamma_0,\dots,\gamma_n$ in $\mathfrak{P}$ from an initial configuration γ_0 to a final configuration γ_n which is labeled with $[\alpha_1\dots\alpha_n] = \sigma_1'\overline{\rho_1'}\dots\sigma_k'\overline{\rho_k'}$. To this end, we define

- $\vec{\boldsymbol{p}}_{\boldsymbol{j}} := \Pi_{\mathrm{rd}(\alpha_1\dots\alpha_j)}$,
- $\vec{\boldsymbol{q}}_{\boldsymbol{j}} := \Pi_{\lambda\mathrm{wrt}(\alpha_1\dots\alpha_j)}$, and
- $c_j := |\lambda_j|_B \geq 0$

Set $\gamma_j := ((s_j, \vec{\boldsymbol{p}}_{\boldsymbol{j}}, \vec{\boldsymbol{q}}_{\boldsymbol{j}}), \#^{c_j}) \in \mathrm{Conf}_{\mathfrak{P}}$. We want to prove now that $\gamma_0,\dots,\gamma_n$ is such run in $\mathfrak{P}$. However, we first note that c_j is the difference of operations from B and $\overline{B}$ in the action sequence $\alpha_1\dots\alpha_j$:

▷ *Claim 1.* *For each $0 \leq j \leq n$ we have $c_j = |\alpha_1\dots\alpha_j|_B - |\alpha_1\dots\alpha_j|_{\overline{B}}$.*

Proof. This proof is very similar to Claim 1 in the proof of Lemma 10.2.15. ◁

The following three statements prove that $\gamma_0,\dots,\gamma_n$ is a run in $\mathfrak{P}$ from an initial state to an accepting state and the j^{th} intermediate content λ_j of the distributed queue is covered by the semantics of the configuration γ_j for any $0 \leq j \leq n$.

▷ *Claim 2.* *We have $\gamma_0 \in \mathrm{Init}_{\mathfrak{P}}$ and $\lambda = \lambda_0 \in [\![\gamma_0]\!]_{\mathfrak{P}}$.*

Proof. We have $s_0 \in I_{\mathfrak{T}}$ by the choice of the run in $\mathfrak{T}$. Additionally, we have $\vec{\boldsymbol{p}}_{\boldsymbol{0}} = \Pi_\varepsilon = \vec{\boldsymbol{\imath}} \in I_{\mathfrak{C}}$ and $\vec{\boldsymbol{q}}_{\boldsymbol{0}} = \Pi_\lambda \in Q_L$ since we have $\vec{\boldsymbol{\imath}} = \vec{\boldsymbol{p}}_{\boldsymbol{0}} \xrightarrow{\lambda}_{\mathfrak{C}} \vec{\boldsymbol{q}}_{\boldsymbol{0}}$ and $\lambda \in L$. We also observe $c_0 = |\lambda_0|_B = |\lambda|_B = 0$ since $\lambda \in L \subseteq \mathbb{M}(\mathcal{A})$. Hence, we infer $\gamma_0 = ((s_0, \vec{\boldsymbol{p}}_{\boldsymbol{0}}, \vec{\boldsymbol{q}}_{\boldsymbol{0}}), \#^0) \in \mathrm{Init}_{\mathfrak{P}}$ and $\lambda \in T(\mathfrak{C}_{\vec{\boldsymbol{p}}_{\boldsymbol{0}} \to \vec{\boldsymbol{q}}_{\boldsymbol{0}}}) \cap \vartheta^{-1}(\#^0) = [\![\gamma_0]\!]_{\mathfrak{P}}$. ◁

▷ *Claim 3.* *For each $0 \leq j < m$ we have $\gamma_j \xrightarrow{\alpha_{j+1}}_{\mathfrak{P}} \gamma_{j+1}$ and $\lambda_{j+1} \in [\![\gamma_{j+1}]\!]_{\mathfrak{P}}$.*

Proof. We prove this by induction on j. By Claim 2 we already know $\lambda_0 \in [\![\gamma_0]\!]_{\mathfrak{P}}$. Hence, we only prove the induction step. To this end, let $j \geq 0$. By the choice of our run in $\mathfrak{T}$ we know $(s_j, \alpha_{j+1}, s_{j+1}) \in \Delta_{\mathfrak{T}}$. Now, we have to take a closer look at $\vec{\boldsymbol{p}}_{\boldsymbol{j+1}}$ and $\vec{\boldsymbol{q}}_{\boldsymbol{j+1}}$. To this end, we distinguish the following two cases:

(W) $\alpha_{j+1} = a \in A'$. Then we have $\vec{\boldsymbol{p}}_{\boldsymbol{j+1}} = \vec{\boldsymbol{p}}_{\boldsymbol{j}}$ and $c_{j+1} = c_j + |a|_B$. Additionally, we have $\vec{\boldsymbol{q}}_{\boldsymbol{j}} = \Pi_{\lambda\mathrm{wrt}(\alpha_1\dots\alpha_j)}$ and $\vec{\boldsymbol{q}}_{\boldsymbol{j+1}} = \Pi_{\lambda\mathrm{wrt}(\alpha_1\dots\alpha_j)a}$. Hence, since Π is an asynchronous run in $\mathfrak{C}$ labeled with $\lambda\sigma_1'\dots\sigma_k' = \lambda\mathrm{wrt}(\alpha_1\dots\alpha_n)$, there is a transition $(\vec{\boldsymbol{q}}_{\boldsymbol{j}}, a, \vec{\boldsymbol{q}}_{\boldsymbol{j+1}}) \in \Delta_{\mathfrak{C}}$. Then by definition our pushdown automaton $\mathfrak{P}$ has the following transition:

$$\big((s_j, \vec{\boldsymbol{p}}_{\boldsymbol{j}}, \vec{\boldsymbol{q}}_{\boldsymbol{j}}), a, \vartheta(a), (s_{j+1}, \vec{\boldsymbol{p}}_{\boldsymbol{j+1}}, \vec{\boldsymbol{q}}_{\boldsymbol{j+1}})\big) \in \Delta_{\mathfrak{P}}$$

(recall that $\vartheta(a) = \varepsilon$ if $a \in A$ or $\vartheta(a) = \#$ if $a \in B$ holds). This implies $\gamma_j \xrightarrow{a}_{\mathfrak{P}} \gamma_{j+1}$. By induction hypothesis we know $\lambda_j \in [\![\gamma_j]\!]_{\mathfrak{P}}$. Then we learn

$$\begin{aligned}
\lambda_{j+1} = [\![a]\!](\lambda_j) = \lambda_j \cdot a &\in [\![\gamma_j]\!]_{\mathfrak{P}} \cdot a \\
&= \big(T(\mathfrak{C}_{\vec{\boldsymbol{p}}_{\boldsymbol{j}} \to \vec{\boldsymbol{q}}_{\boldsymbol{j}}}) \cap \vartheta^{-1}(\#^{c_j})\big) \cdot a \\
&\subseteq \big(T(\mathfrak{C}_{\vec{\boldsymbol{p}}_{\boldsymbol{j}} \to \vec{\boldsymbol{q}}_{\boldsymbol{j}}}) \cap \vartheta^{-1}(\#^{c_j})\big) \cdot \big(T(\mathfrak{C}_{\vec{\boldsymbol{q}}_{\boldsymbol{j}} \to \vec{\boldsymbol{q}}_{\boldsymbol{j+1}}}) \cap \vartheta^{-1}(\#^{|a|_B})\big) \\
&\subseteq \big(T(\mathfrak{C}_{\vec{\boldsymbol{p}}_{\boldsymbol{j}} \to \vec{\boldsymbol{q}}_{\boldsymbol{j}}}) \cdot T(\mathfrak{C}_{\vec{\boldsymbol{q}}_{\boldsymbol{j}} \to \vec{\boldsymbol{q}}_{\boldsymbol{j+1}}})\big) \cap \vartheta^{-1}(\#^{c_j + |a|_B}) \\
&\subseteq T(\mathfrak{C}_{\vec{\boldsymbol{p}}_{\boldsymbol{j+1}} \to \vec{\boldsymbol{q}}_{\boldsymbol{j+1}}}) \cap \vartheta^{-1}(\#^{c_{j+1}}) = [\![\gamma_{j+1}]\!]_{\mathfrak{P}}\,.
\end{aligned}$$

(R) $\alpha_{j+1} = \overline{a} \in \overline{A'}$. Then we have $\vec{\boldsymbol{q}}_{\boldsymbol{j+1}} = \vec{\boldsymbol{q}}_{\boldsymbol{j}}$ and $c_{j+1} = c_j - |\overline{a}|_{\overline{B}} \geq 1$. Since Π is an asynchronous run in $\mathfrak{C}$ labeled with $\lambda\sigma'_1 \dots \sigma'_k = \lambda\mathrm{wrt}(\alpha_1 \dots \alpha_n)$, $\lambda_j = a\lambda_{j+1}$ holds (by $[\![\overline{a}]\!](\lambda_j) = \lambda_{j+1}$), and since $\vec{\boldsymbol{p}}_{\boldsymbol{j}} \xrightarrow{\lambda_j}_{\mathfrak{C}} \vec{\boldsymbol{q}}_{\boldsymbol{j}}$ there is a unique state $\Pi_{\mathrm{rd}(\alpha_1 \dots \alpha_j)a}$ with

$$\left(\vec{\boldsymbol{p}}_{\boldsymbol{j}} \xrightarrow{\lambda_j}_{\mathfrak{C}} \vec{\boldsymbol{q}}_{\boldsymbol{j}}\right) = \left(\vec{\boldsymbol{p}}_{\boldsymbol{j}} = \Pi_{\mathrm{rd}(\alpha_1 \dots \alpha_j)} \xrightarrow{a}_{\mathfrak{C}} \Pi_{\mathrm{rd}(\alpha_1 \dots \alpha_j)a} \xrightarrow{\lambda_{j+1}}_{\mathfrak{C}} \vec{\boldsymbol{q}}_{\boldsymbol{j}} = \vec{\boldsymbol{q}}_{\boldsymbol{j+1}}\right).$$

Note that we have $\vec{\boldsymbol{p}}_{\boldsymbol{j+1}} = \Pi_{\mathrm{rd}(\alpha_1 \dots \alpha_j)a}$ by definition. Hence, we learn $(\vec{\boldsymbol{p}}_{\boldsymbol{j}}, a, \vec{\boldsymbol{p}}_{\boldsymbol{j+1}}) \in \Delta_{\mathfrak{C}}$. To prove the existence of the transition

$$\left((s_j, \vec{\boldsymbol{p}}_{\boldsymbol{j}}, \vec{\boldsymbol{q}}_{\boldsymbol{j}}), \overline{a}, \overline{\vartheta(a)}, (s_{j+1}, \vec{\boldsymbol{p}}_{\boldsymbol{j+1}}, \vec{\boldsymbol{q}}_{\boldsymbol{j+1}})\right) \in \Delta_{\mathfrak{P}} \tag{10.3}$$

we still have to show $T(\mathfrak{C}_{\vec{\boldsymbol{p}}_{\boldsymbol{j+1}} \to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A}) \neq \varnothing$.

Recall that the read actions we apply on our run $s_0, \dots, s_n$ are the following sequence:

$$\mathrm{rd}(\alpha_1 \dots \alpha_n) = w'_{1,1} \dots w'_{k+1,1} \dots w'_{1,k} \dots w'_{k+1,k}\,.$$

Hence, there are two indices $1 \leq g \leq k+1$ and $1 \leq h \leq k$ and two words $u, v \in A'^*$ satisfying the following properties (in the free monoid A'^*):

(i) $w'_{g,h} = uav$ (recall that $\alpha_{j+1} = \overline{a}$ holds),

(ii) $\mathrm{rd}(\alpha_1 \dots \alpha_j) = w'_{1,1} \dots w'_{g-1,h}u$, and

(iii) $\mathrm{rd}(\alpha_{j+2} \dots \alpha_n) = vw'_{g+1,h} \dots w'_{k+1,k}$.

In other words, before applying α_{j+1} to λ_j we already read u from $w'_{g,h}$, now we read this particular letter a, and afterwards we will read v and the remaining $w'_{x,y}$'s. The trace

$$\left[\mathrm{rd}(\alpha_{j+2} \dots \alpha_n)\right] \cdot \kappa' = \left[vw'_{g+1,h} \dots w'_{k+1,k}\, w'_{1,k+1} \dots w'_{k+1,k+1}\right]$$

is the complementary suffix of $\lambda\sigma'_1 \dots \sigma'_k$ wrt. $\mathrm{rd}(\alpha_1 \dots \alpha_{j+1})$ (in the trace monoid $\mathbb{M}(\mathcal{A}')$) implying $\vec{\boldsymbol{p}}_{\boldsymbol{j}} \xrightarrow{[vw'_{g+1,h} \dots w'_{k+1,k+1}]}_{\mathfrak{C}} F_{\mathfrak{C}}$. We now want to find another run in $\mathfrak{C}$ from $\vec{\boldsymbol{p}}_{\boldsymbol{j}}$ to $F_{\mathfrak{C}}$ which is labeled by a trace containing no begin markers. To this end, we will prune some W'-factors of $vw'_{g+1,h} \dots w'_{k+1,k+1}$ (recall that each of these W'-factors start with a letter from B). First, with the help of our special variation of Levi's lemma (Corollary 10.4.7(5)) we can see that the following equation holds (cf. Figure 10.7):

$$vw'_{g+1,h} \dots w'_{k+1,k+1} \approx_{\mathcal{A}'} v \cdot \prod_{1 \leq y \leq g} (w'_{y,h+1} \dots w'_{y,k+1}) \cdot \prod_{g < y \leq k+1} (w'_{y,h} \dots w'_{y,k+1})\,. \tag{10.4}$$

Let $1 \leq x \leq k+1$ be an index such that $w'_{x,h+1} \dots w'_{x,k+1}$ (or $w'_{x,h} \dots w'_{x,k+1}$ if $x > g$) contains a begin marker $\$_{C_x,i_x} \in B$. Since each row starts with its begin marker $\$_{C_x,i_x}$ according to our construction of the words $w'_{y,z}$, we obtain $w'_{x,h+1} \dots w'_{x,k+1} \in W_{C_x,i_x} \subseteq W'$. Then we see that the word

$$u_x := v \cdot \prod_{\substack{1 \leq y \leq g \\ y \neq x}} (w'_{y,h+1} \dots w'_{y,k+1}) \cdot \prod_{\substack{g < y \leq k+1 \\ y \neq x}} (w'_{y,h} \dots w'_{y,k+1})$$

is still the complementary suffix of $\mathrm{rd}_i(\alpha_1 \dots \alpha_j)$ wrt. a trace in LW'^*. Hence, we learn $[u_x] \in T(\mathfrak{C}_{\vec{\boldsymbol{p}}_{\boldsymbol{j}} \to F_{\mathfrak{C}}})$ and has one occurrence of B fewer than the trace $[vw'_{g+1,h} \dots w'_{k+1,k+1}]$. We can iterate this pruning of factors $w'_{x,h+1} \dots w'_{x,k+1}$ which results in a trace from

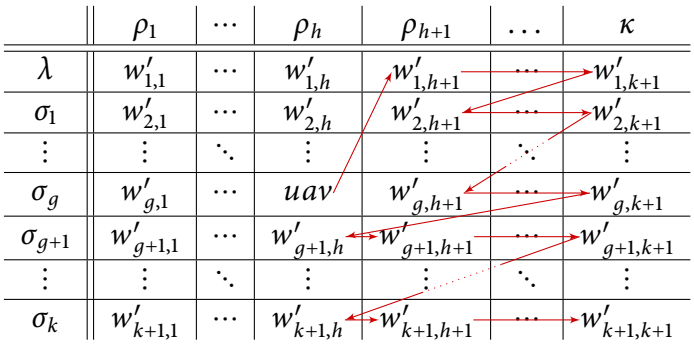

Figure 10.7. Visualization of Equation (10.4): The left-hand side of this equation can be read column-wise from starting from the cell containing uav. The right-hand side of this equation can be read row-wise as depicted by the arrows.

$T(\mathfrak{C}_{\vec{p}_{j+1}\to F_{\mathfrak{C}}})$ containing letters from A, only. In other words, we have $T(\mathfrak{C}_{\vec{p}_{j+1}\to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A}) \neq \varnothing$.

We infer the existence of the transition in Equation (10.3) implying $\gamma_j \xrightarrow{\overline{a}}_{\mathfrak{P}} \gamma_{j+1}$. Furthermore, by the induction hypothesis we know

$$a\lambda_{j+1} = \lambda_j \in [\![\gamma_j]\!]_{\mathfrak{P}} = T(\mathfrak{C}_{\vec{p}_j\to\vec{q}_j}) \cap \vartheta^{-1}(\#^{c_j}) .$$

We learn $\lambda_{j+1} \in T(\mathfrak{C}_{\vec{p}_{j+1}\to\vec{q}_{j+1}})$ and $\lambda_{j+1} \in \vartheta^{-1}(\#^{c_j - |\overline{a}|_{\overline{B}}}) = \vartheta^{-1}(\#^{c_{j+1}})$. This implies $\lambda_{j+1} \in [\![\gamma_{j+1}]\!]_{\mathfrak{P}}$. ◁

▷ *Claim 4.* *We have* $\gamma_n \in \mathrm{Final}_{\mathfrak{P}}$.

Proof. We know $\vec{q}_n = \Pi_{\lambda \mathrm{wrt}(\alpha_1\dots\alpha_n)} = \Pi_{\lambda\sigma'_1\dots\sigma'_k} = \vec{f} \in F_{\mathfrak{C}}$ and $s_n \in F_{\mathfrak{T}}$. We infer $(s_n, \vec{p}_n, \vec{q}_n) \in F_{\mathfrak{P}}$ and, hence, $\gamma_n \in \mathrm{Final}_{\mathfrak{P}}$. ◁

All in all, we obtain $\kappa' = [\![\sigma'_1\overline{\rho'_1}\dots\sigma'_k\overline{\rho'_k}]\!](\lambda) = \lambda_n \in [\![\gamma_n]\!]_{\mathfrak{P}}$ and, hence, $[\![\tau]\!](\lambda) = \kappa = \pi_A(\kappa') \in \pi_A([\![\gamma_n]\!]_{\mathfrak{P}})$. Additionally, we know $\gamma_n \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$ which finally proves our lemma. ◂

Now, we prove the converse inclusion of Proposition 10.4.10. The proof of this inclusion proceeds similar to the proof of Lemma 10.2.17. So, we consider a τ-labeled, accepting run in $\mathfrak{P}$ to a final configuration $\gamma \in \mathrm{Conf}_{\mathfrak{P}}$ and a $\kappa \in [\![\gamma]\!]_{\mathfrak{P}}$. Similar to the single-queue case our constructed pushdown automaton $\mathfrak{P}$ lacks a memory of letters we have written before. Hence, it is possible that there is no $\lambda \in L$ with $[\![\tau]\!](\lambda) = \kappa$. However, we can exploit that lack of memory and construct another action trace $\tau' \in (W'\overline{R'})^*$ from λ and τ such that $[\![\tau']\!](\lambda) = \kappa$. We do so by replacing the write actions in τ by the letters we read in τ. This ensures that we always read only those letters from the distributed queue's contents after we have written them into its content.

Lemma 10.4.12. *Let* $\gamma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$ *be a reachable, accepting configuration. Then we have* $\pi_A([\![\gamma]\!]_{\mathfrak{P}}) \subseteq \textsc{Reach}(L, (W\overline{R})^*)$.

Proof. Since $\gamma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$ there is a run $(\gamma_j)_{0\le j\le n} = ((s_j, \vec{\boldsymbol{p}}_j, \vec{\boldsymbol{q}}_j), \#^{c_j})_{0\le j\le n}$ in $\mathfrak{P}$ with $\gamma_0 \in \mathrm{Init}_{\mathfrak{P}} = (I_{\mathfrak{T}} \times I_{\mathfrak{C}} \times Q_L) \times \{\varepsilon\}$ and $\gamma_n = \gamma \in \mathrm{Final}_{\mathfrak{P}} = (F_{\mathfrak{T}} \times \vec{\boldsymbol{Q}}_{\mathfrak{C}} \times F_{\mathfrak{C}}) \times \#^*$. Then there are basic queue actions $\alpha_1, \dots, \alpha_n \in \Sigma'$ with $\gamma_j \xrightarrow{\alpha_{j+1}}_{\mathfrak{P}} \gamma_{j+1}$ for each $0 \le j < n$. By the definition of $\mathfrak{P}$ we know the following for $0 \le j < n$:

(i) $(s_j, \alpha_{j+1}, s_{j+1}) \in \Delta_{\mathfrak{T}}$. Hence we observe $[\alpha_1 \dots \alpha_n] \in T(\mathfrak{T})^* = (W'\overline{R'})^*$. Then there are traces $\sigma_1, \dots, \sigma_k \in W'$ and $\rho_1, \dots, \rho_k \in R'$ with $[\alpha_1 \dots \alpha_n] = \sigma_1\overline{\rho_1} \dots \sigma_k\overline{\rho_k}$.

(ii) $(\vec{\boldsymbol{p}}_j, a, \vec{\boldsymbol{p}}_{j+1}) \in \Delta_{\mathfrak{C}}$ if $\alpha_{j+1} = \overline{a} \in \overline{A'}$ or $\vec{\boldsymbol{p}}_j = \vec{\boldsymbol{p}}_{j+1}$ if $\alpha_{j+1} \in A'$. Therefore, we have

$$\rho_1 \dots \rho_k = [\mathrm{rd}(\alpha_1 \dots \alpha_n)] \in T(\mathfrak{C}_{I_{\mathfrak{C}} \to \vec{\boldsymbol{p}}_n}) .$$

Additionally, we have $T(\mathfrak{C}_{\vec{\boldsymbol{p}}_{j+1} \to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A}) \neq \varnothing$ by the definition of $\mathfrak{P}$.

(iii) $(\vec{\boldsymbol{q}}_j, a, \vec{\boldsymbol{q}}_{j+1}) \in \Delta_{\mathfrak{C}}$ if $\alpha_{j+1} = a \in A'$ or $\vec{\boldsymbol{q}}_j = \vec{\boldsymbol{q}}_{j+1}$ if $\alpha_{j+1} \in \overline{A'}$. From $\gamma_0 \in \mathrm{Init}_{\mathfrak{P}}$ we learn $\vec{\boldsymbol{q}}_0 \in Q_L$ implying

$$\sigma_1 \dots \sigma_k = [\mathrm{wrt}(\alpha_1 \dots \alpha_n)] \in T(\mathfrak{C}_{\vec{\boldsymbol{q}}_0 \to \vec{\boldsymbol{q}}_n}) \subseteq T(\mathfrak{C}_{Q_L \to F_{\mathfrak{C}}}) = W'^* .$$

Now, let $\kappa \in [\![\gamma_n]\!]_{\mathfrak{P}} = T(\mathfrak{C}_{\vec{\boldsymbol{p}}_n \to \vec{\boldsymbol{q}}_n}) \cap \vartheta^{-1}(\#^{c_n})$. Then we have

$$\begin{aligned} \rho_1 \dots \rho_k \cdot \kappa &\in T(\mathfrak{C}_{I_{\mathfrak{C}} \to \vec{\boldsymbol{p}}_n}) \cdot T(\mathfrak{C}_{\vec{\boldsymbol{p}}_n \to \vec{\boldsymbol{q}}_n}) \\ &\subseteq T(\mathfrak{C}) = LW'^* . && \text{(by } \gamma_n \in \mathrm{Final}_{\mathfrak{P}} \text{ and } \vec{\boldsymbol{q}}_n \in F_{\mathfrak{C}}) \end{aligned}$$

Hence, there are $\lambda \in L$ and $\sigma'_1, \dots, \sigma'_x \in W'$ with $\rho_1 \dots \rho_k\kappa = \lambda\sigma'_1 \dots \sigma'_x$.
▷ *Claim 1.* *We have $k = x$.*
Proof. First, we obtain the following equation:

$$|\rho_1 \dots \rho_k|_B + |\kappa|_B = |\rho_1 \dots \rho_k\kappa|_B = |\lambda\sigma'_1 \dots \sigma'_x|_B = x \tag{10.5}$$

since $|\lambda|_B = 0$ and $|\sigma'_j|_B = 1$ for each $1 \le j \le x$. Additionally, since the stack of $\mathfrak{P}$ always contains the difference of letters from B and $\overline{B}$ seen before, we have

$$|\kappa|_B = c_n = |\sigma_1 \dots \sigma_k|_B - |\rho_1 \dots \rho_k|_B = k - |\rho_1 \dots \rho_k|_B . \tag{10.6}$$

Combining the Equations (10.5) and (10.6) results in $k = x$. ◁

For $1 \le j \le k$ let $C_j \subseteq A$ and $i_j \in P \cup \{0\}$ with $\sigma'_j \in W_{C_j,i_j}$. Let $1 \le i < j \le k$ with $\$_{C_i,i_i}\$_{C_j,i_j} \approx_{\mathcal{A}'} \$_{C_j,i_j}\$_{C_i,i_i}$. Then we obtain $C_i \parallel C_j$. We also know $C_i = \mathrm{Alph}(\sigma'_i) \setminus \{\$_{C_i,i_i}\}$ and $C_j = \mathrm{Alph}(\sigma'_j) \setminus \{\$_{C_j,i_j}\}$ implying $\sigma'_i \parallel \sigma'_j$. We infer

$$\sigma'_i\sigma'_j = \sigma'_j\sigma'_i \tag{10.7}$$

in this case. Hence, we can assume that $\overline{\$_{C_1,i_1} \dots \$_{C_k,i_k}} = \pi_{\overline{B}}(\alpha_1 \dots \alpha_n)$ holds (in the free monoid $\overline{B}^*$).

We also have to determine the number of letters from B in each trace $\rho_1 \dots \rho_j$ (where $1 \le j \le k$). Similar to Equation (10.6) we know $c_i = |\alpha_1 \dots \alpha_i|_B - |\alpha_1 \dots \alpha_i|_{\overline{B}} \ge 0$ for each $1 \le i \le n$. This implies for $1 \le j \le k$

$$|\rho_1 \dots \rho_j|_B \le |\sigma_1 \dots \sigma_j|_B = j = |\sigma'_1 \dots \sigma'_j|_B . \tag{10.8}$$

Next, we want to construct traces $\mu_{i,j} \in \mathbb{M}(\mathcal{A}')$ with $1 \leq i, j \leq k+1$ satisfying the properties (1)-(6) of our special variation of Levi's lemma (Corollary 10.4.7). Application of this corollary will finally yield $\kappa = [\![\sigma_1'\overline{\rho_1} \dots \sigma_k'\overline{\rho_k}]\!](\lambda)$ and, hence, $\pi_A(\kappa) \in \textsc{Reach}(L, (W\overline{R})^*)$. The construction of the $\mu_{i,j}$ is done inductively. So, we construct traces $v_{i,1}^{(j)}$ and $v_{i,2}^{(j)}$ satisfying the original Levi's lemma (cf. Theorem 3.4.6) for an increasing number j. Intuitively, we fill the table in Figure 3.3 column-wise, where $v_{i,1}^{(j)}$ corresponds to the cell (i, j) and $v_{i,2}^{(j)}$ is the concatenation of all columns to the right of the j^{th} column (cf. Figure 10.8).

	ρ_1	ρ_2	$\cdots$	ρ_j	$\rho_{j+1}\cdots\rho_k\kappa$
λ	$v_{1,1}^{(1)}$	$v_{1,1}^{(2)}$	$\cdots$	$v_{1,1}^{(j)}$	$v_{1,2}^{(j)}$
σ_1'	$v_{2,1}^{(1)}$	$v_{2,1}^{(2)}$	$\cdots$	$v_{2,1}^{(j)}$	$v_{2,2}^{(j)}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
σ_k'	$v_{k+1,1}^{(1)}$	$v_{k+1,1}^{(2)}$	$\cdots$	$v_{k+1,1}^{(j)}$	$v_{k+1,2}^{(j)}$

Figure 10.8

First, we set $v_{1,2}^{(0)} := \lambda$ and $v_{i+1,2}^{(0)} := \sigma_i'$ for each $1 \leq i \leq k$. Then we know

$$v_{1,2}^{(0)} v_{2,2}^{(0)} \dots v_{k+1,2}^{(0)} = \lambda\sigma_1' \dots \sigma_k' = \rho_1 \dots \rho_k\kappa\,.$$

The induction step proceeds as follows: for $1 \leq j \leq k$ we know $v_{i,2}^{(j-1)} \dots v_{k+1,2}^{(j-1)} = \rho_j \cdot (\rho_{j+1} \cdot \rho_k\kappa)$ by induction hypothesis. By Levi's lemma (Theorem 3.4.6) there are traces $v_{i,1}^{(j)}, v_{i,2}^{(j)} \in \mathbb{M}(\mathcal{A}')$ with the following properties:

(1) $v_{i,2}^{(j-1)} = v_{i,1}^{(j)} v_{i,2}^{(j)}$ for each $1 \leq i \leq k+1$,

(2) $\rho_j = v_{1,1}^{(j)} \dots v_{k+1,1}^{(j)}$,

(3) $\rho_{j+1} \cdot \rho_k\kappa = v_{1,2}^{(j)} \dots v_{k+1,2}^{(j)}$, and

(4) $v_{g,1}^{(j)} \parallel v_{h,2}^{(j)}$ for each $g > h$.

Now, that we have constructed the traces $v_{i,1}^{(j)}$ and $v_{i,2}^{(j)}$, we have to prove that the lower-left triangle of the table in Figure 10.8 is empty:

▷ *Claim 2.* *We have $v_{i,1}^{(j)} = \varepsilon$ for all $i > j+1$.*

Proof. Set $\ell := |\sigma_1\overline{\rho_1} \dots \sigma_j\overline{\rho_j}|$ (i.e., our pushdown automaton is in configuration γ_ℓ after application of the j^{th} alternation). Recall that we have $T(\mathfrak{C}_{\vec{p}_\ell \to F_\mathfrak{C}}) \cap \mathbb{M}(\mathcal{A}) \neq \varnothing$ by property (ii). Hence, there is another trace $\xi \in T(\mathfrak{C}_{\vec{p}_\ell \to F_\mathfrak{C}}) \cap \mathbb{M}(\mathcal{A})$. By $\rho_1 \dots \rho_j \in T(\mathfrak{C}_{I_\mathfrak{C} \to \vec{p}_\ell})$ we obtain

$$\rho_1 \dots \rho_j \cdot \xi \in T(\mathfrak{C}_{I_\mathfrak{C} \to \vec{p}_\ell}) \cdot T(\mathfrak{C}_{\vec{p}_\ell \to F_\mathfrak{C}}) \subseteq T(\mathfrak{C}) = LW'^*\,.$$

Additionally, we already know $|\rho_1 \dots \rho_j|_B \leq j$ from Equation (10.8) implying the existence of a number $g \leq j$ such that $\rho_1 \dots \rho_j\xi \in LW'^g$ holds, i.e., there are traces $\lambda' \in L$ and $\sigma_1'', \dots, \sigma_g'' \in W'$ with $\rho_1 \dots \rho_j\xi = \lambda'\sigma_1'' \dots \sigma_g''$. Again, we can apply Levi's lemma (Theorem 3.4.6) which results in the table in Figure 10.9.

In other words, we replace the column "$\rho_{j+1} \dots \rho_k\kappa$" in Figure 10.8 by ξ. Additionally, since $g \leq j \leq k$ holds, at most the first and g further rows are non-empty. Concretely, we observe

	ρ_1	$\cdots$	ρ_j	$\rho_{j+1} \dots \rho_k \kappa \rightsquigarrow \xi$
$\lambda \rightsquigarrow \lambda'$	$v_{1,1}^{(1)}$	$\cdots$	$v_{1,1}^{(j)}$	
$\sigma_1' \rightsquigarrow \sigma_1''$	$v_{2,1}^{(1)}$	$\cdots$	$v_{2,1}^{(j)}$	
$\vdots$	$\vdots$	$\ddots$	$\vdots$	
$\sigma_g' \rightsquigarrow \sigma_g''$	$v_{g,1}^{(1)}$	$\cdots$	$v_{g,1}^{(j)}$	ξ
$\sigma_{g+1}' \rightsquigarrow \varepsilon$	$v_{g+1,1}^{(1)}$	$\cdots$	$v_{g+1,1}^{(j)}$	
$\vdots$	$\vdots$	$\ddots$	$\vdots$	
$\sigma_k' \rightsquigarrow \varepsilon$	$v_{k+1,1}^{(1)}$	$\cdots$	$v_{k+1,1}^{(j)}$	

Figure 10.9. Visualization of the equation $\rho_1 \dots \rho_j \xi = \lambda' \sigma_1'' \dots \sigma_g''$ in terms of the traces $v_{i,1}^{(j)}$.

that λ and λ' have the common prefix $v_{1,1}^{(1)} \dots v_{1,1}^{(j)}$. We can also assume that σ_i' and σ_i'' have the common prefix $v_{i+1,1}^{(1)} \dots v_{i+1,1}^{(j)}$ for each $1 \leq i \leq g$. Note that in the other case we could observe $\$_{C_x,i_x}\$_{C_y,i_y} \approx_{\mathcal{A}'} \$_{C_y,i_y}\$_{C_x,i_x}$ implying $\sigma_x'\sigma_y' = \sigma_y'\sigma_x'$ according to Equation (10.7). This implies that the remaining rows $g < i \leq k$ remain empty, i.e., we have $v_{i+1,1}^{(j)} = \varepsilon$ for each $g < j+1 \leq i \leq k$. ◁

Now, we set $\mu_{i,j} := v_{i,1}^{(j)}$ for each $1 \leq i \leq k+1$ and $1 \leq j \leq k$. Additionally, we set $\mu_{i,k+1} := v_{i,2}^{(k)}$ for each $1 \leq i \leq k+1$. Then we already know that the traces $\mu_{i,j}$ satisfy property (6) of Corollary 10.4.7. Similar to the proof of Corollary 10.4.7 we may also show the correctness of the remaining properties (1)-(5).

Finally, the application of Corollary 10.4.7 results in $\kappa = [\![\sigma_1'\overline{\rho_1} \dots \sigma_k'\overline{\rho_k}]\!](\lambda)$, i.e., we have $\kappa \in \textsc{Reach}(L, (W'\overline{R'})^*)$. Due to our construction of $\mathcal{A}'$, W', and R' we also have $\pi_A(\kappa) \in \textsc{Reach}(L, (W\overline{R})^*)$. Since $\kappa \in [\![\gamma_n]\!]_{\mathfrak{P}}$ was arbitrary, we learn

$$\pi_A([\![\gamma_n]\!]_{\mathfrak{P}}) \subseteq \textsc{Reach}(L, (W\overline{R})^*)\,. \qquad \blacktriangleleft$$

Finally, Lemmata 10.4.11 and 10.4.12 prove Proposition 10.4.10.

Correctness of the Construction

Before we prove the correctness of our construction, we need another lemma. This one proves that we can factorize the semantics of any reachable accepting configuration $\gamma = ((s, \vec{\boldsymbol{p}}, \vec{\boldsymbol{q}}), \#^c)$ of $\mathfrak{P}$ into three recognizable languages: (1) some letters from A which belong to traces $\sigma \in L \cup W'$ from which we have already read several letters (especially we have already read the begin markers of the traces σ), (2) $c-1$ traces from W' (i.e., W'^{c-1}), and (3) one further trace from W' which has a run in $\mathfrak{C}$ ending in $\vec{\boldsymbol{q}} \in F_{\mathfrak{C}}$ (note that in contrast to the single-queue case we could not assume $|F_{\mathfrak{C}}| = 1$ in this case).

Lemma 10.4.13. *Let* $\gamma = ((s, \vec{\boldsymbol{p}}, \vec{\boldsymbol{q}}), \#^c) \in \text{post}^*_{\mathfrak{P}}(\text{Init}_{\mathfrak{P}}) \cap \text{Final}_{\mathfrak{P}}$ *with* $c > 0$ *and let* $G := \{\vec{\boldsymbol{r}} \in \vec{\boldsymbol{Q}}_{\mathfrak{C}} \mid \exists \lambda \in LW'^*\colon I_{\mathfrak{C}} \xrightarrow{\lambda}_{\mathfrak{C}} \vec{\boldsymbol{r}}\}$ *(note that* $F_{\mathfrak{C}} \subseteq G$ *holds). Then we have*

$$[\![\gamma]\!]_{\mathfrak{P}} = (T(\mathfrak{C}_{\vec{\boldsymbol{p}} \to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A})) \cdot W'^{c-1} \cdot (T(\mathfrak{C}_{G \to \vec{\boldsymbol{q}}}) \cap W')\,.$$

With the help of this lemma we can also prove the recognizability of the semantics of a regular set of configurations of $\mathfrak{P}$. To do this, we only have to replace the second factor "W'^{c-1}" by "W'^{N}" where $N \subseteq \mathbb{N}$ is recognizable in $\mathbb{N}$. Note that W'^{N} also is efficiently recognizable in $\mathbb{M}(\mathcal{A}')$ by the efficient closure properties (cf. Appendix A).

Proof. First, let $\kappa \in [\![\gamma]\!]_{\mathfrak{P}}$. By Proposition 10.4.10 there are $\lambda \in L$, $\sigma_1, \dots, \sigma_k \in W'$, and $\rho_1, \dots, \rho_k \in R'$ with $[\![\sigma_1\overline{\rho_1} \dots \sigma_k\overline{\rho_k}]\!](\lambda) = \kappa \neq \bot$. Our modified version of Levi's lemma (Corollary 10.4.7) yields traces $\mu_{i,j} \in \mathbb{M}(\mathcal{A}')$ for $1 \leq i, j \leq k+1$ satisfying the properties (1)-(6). We set $X := \{1 \leq j \leq k \mid \mu_{j+1,k+1} \neq \sigma_j\}$, i.e., X is the set of indices j of σ_j from which we have read at least one letter and, hence, which are not fully contained in κ.

In our first claim we consider a trace σ_j from which we have read at least one letter. We will show that we have particularly read its begin marker. Formally, the complementary prefix of σ_j wrt. $\mu_{j+1,k+1}$ starts with a letter from B.

▷ *Claim 1. Let $j \in X$. Then we have $\mu_{j+1,1} \dots \mu_{j+1,k} \in \zeta_{\mathcal{A}'}(BA^*)$.*

Proof. Towards a contradiction, we suppose $\mu_{j+1,1} \dots \mu_{j+1,k} \notin \zeta_{\mathcal{A}'}(BA^*)$, i.e., this trace does not contain a letter from B. However, by $j \in X$ we know $\mu_{j+1,k+1} \neq \sigma_j$ implying $\mu_{j+1,1} \dots \mu_{j+1,k} \neq \varepsilon$.

From $\gamma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}})$ we know $T(\mathfrak{C}_{\vec{p} \to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A}) \neq \varnothing$. Hence, there is a trace $\xi \in T(\mathfrak{C}_{\vec{p} \to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A})$. We also know $\rho_1 \dots \rho_k \in T(\mathfrak{C}_{I_{\mathfrak{C}} \to \vec{p}})$ by the construction of our pushdown system $\mathfrak{P}$. We infer

$$\rho_1 \dots \rho_k \cdot \xi \in T(\mathfrak{C}_{I_{\mathfrak{C}} \to \vec{p}}) \cdot T(\mathfrak{C}_{\vec{p} \to F_{\mathfrak{C}}}) \subseteq T(\mathfrak{C}) = LW'^{*}\,.$$

Now, we replace the last row of the table in Figure 10.4 by ξ. In other words, we find traces $\nu_{x,k+1} \in \mathbb{M}(\mathcal{A})$ with the following properties: (1) $\nu_{1,k+1} \dots \nu_{k+1,k+1} = \xi$, (2) $\mu_{1,1} \dots \mu_{1,k}\nu_{1,k+1} \in L$, and (3) $\mu_{x+1,1} \dots \mu_{x+1,k}\nu_{x+1,k+1} \in W' \cup \{\varepsilon\}$ for each $1 \leq x \leq k$.

Though, from our assumption $\mu_{j+1,1} \dots \mu_{j+1,k} \notin \zeta_{\mathcal{A}'}(BA^*)$ and $\nu_{j+1,k+1} \in \mathbb{M}(\mathcal{A})$ we learn

$$\varepsilon \neq \mu_{j+1,1} \dots \mu_{j+1,k}\nu_{j+1,k+1} \in W' \cap \mathbb{M}(\mathcal{A}) = \varnothing\,,$$

but this is impossible. Hence, we have $\mu_{j+1,1} \dots \mu_{j+1,k} \in \zeta_{\mathcal{A}'}(BA^*)$. ◁

From this claim we learn that $\mu_{j+1,k+1} \in \mathbb{M}(\mathcal{A})$ holds if, and only if, $j \in X$. This implies

$$c = |\sigma_1 \dots \sigma_k|_B - |\rho_1 \dots \rho_k|_B = k - |\mu_{1,1} \dots \mu_{k+1,1} \dots \mu_{1,k} \dots \mu_{k+1,k}|_B = k - |X|\,.$$

▷ *Claim 2. Let $i \in X$ and $1 \leq j < i$ with $j \notin X$. Then we have $\sigma_i\sigma_j = \sigma_j\sigma_i$.*

Proof. Let $C_i, C_j \subseteq A$ and $i_i, i_j \in P \cup \{0\}$ with $\sigma_i \in W_{C_i,i_i}$ and $\sigma_j \in W_{C_j,i_j}$. By $j \notin X$ we know $\mu_{j+1,1} \dots \mu_{j+1,k} = \varepsilon$ and $\mu_{j+1,k+1} = \sigma_j$. From $i \in X$ we also infer $\mu_{i+1,1} \dots \mu_{i+1,k} \in \zeta_{\mathcal{A}'}(\$_{C_i,i_i}A^*)$. Since $j < i$ the automaton $\mathfrak{P}$ has written σ_j before σ_i, but it has read $\$_{C_i,i_i}$ before σ_j. Hence, we obtain $\$_{C_i,i_i}\$_{C_j,i_j} \approx_{\mathcal{A}'} \$_{C_j,i_j}\$_{C_i,i_i}$. This equation implies one of the following two cases:

(1) $\$_{C_i,i_i} = \$_{C_j,i_j}$. From property (5) of Corollary 10.4.7 we infer $\mu_{i+1,1} \dots \mu_{i+1,k} \parallel \mu_{j+1,k+1}$. But this is impossible since $\$_{C_i,i_i} \in \mathrm{Alph}(\mu_{i+1,1} \dots \mu_{i+1,k}) \cap \mathrm{Alph}(\mu_{j+1,k+1})$ and, by definition of distributed alphabets, we have $\$_{C_i,i_i}\,\mathrm{M}' \neq \varnothing$. Hence, this case will not occur.

(2) $\$_{C_i,i_i} \parallel \$_{C_j,i_j}$. By the construction of $\mathcal{A}'$ we also have $C_i \parallel C_j$ in this case. Since $C_i = \mathrm{Alph}(\sigma_i) \setminus \{\$_{C_i,i_i}\}$ and $C_j = \mathrm{Alph}(\sigma_j) \setminus \{\$_{C_j,i_j}\}$ holds, we finally obtain $\sigma_i\sigma_j = \sigma_j\sigma_i$. ◁

In particular, from this claim we infer $\mu_{i+1,k+1}\mu_{j+1,k+1} = \mu_{j+1,k+1}\mu_{i+1,k+1}$ for $i \in X$ and $j \notin X$ with $1 \leq j < i \leq k$. Therefore we obtain

$$\kappa = \prod_{j \in X \cup \{0\}} \mu_{j+1,k+1} \cdot \prod_{j \notin X \cup \{0\}} \mu_{j+1,k+1} = \prod_{j \in X \cup \{0\}} \mu_{j+1,k+1} \cdot \prod_{j \notin X \cup \{0\}} \sigma_j\,.$$

By the definition of $\mathfrak{P}$ and by $\gamma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}})$ we know

$$\prod_{j\in X\cup\{0\}} \mu_{j+1,1}\dots\mu_{j+1,k} \in T(\mathfrak{C}_{I_{\mathfrak{C}}\to\vec{\boldsymbol{p}}}) \quad \text{and} \quad \prod_{j\in X\cup\{0\}} \mu_{j+1,k+1} \in T(\mathfrak{C}_{\vec{\boldsymbol{p}}\to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A})\,.$$

Additionally, we have $\prod_{j\notin X\cup\{0\}} \mu_{j+1,k+1} \in W'^c$. Since $\kappa \in T(\mathfrak{C}_{\vec{\boldsymbol{p}}\to\vec{\boldsymbol{q}}})$ holds, we finally infer

$$\kappa \in \left(T(\mathfrak{C}_{\vec{\boldsymbol{p}}\to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A})\right)\cdot W'^{c-1}\cdot\left(T(\mathfrak{C}_{G\to\vec{\boldsymbol{q}}}) \cap W'\right).$$

Next, we prove the converse inclusion. To this end, let $\kappa \in \left(T(\mathfrak{C}_{\vec{\boldsymbol{p}}\to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A})\right)\cdot W'^{c-1}\cdot\left(T(\mathfrak{C}_{G\to\vec{\boldsymbol{q}}}) \cap W'\right)$. We see then that $\kappa \in \pi_B^{-1}(B^c) = \vartheta^{-1}(\#^c)$ holds. Additionally, we know from $F_{\mathfrak{C}} \subseteq G$ that $W' = T(\mathfrak{C}_{G\to G}) \cap \zeta_{\mathcal{A}'}(BA^*)$ holds. Hence, there are states $\vec{\boldsymbol{f}}_1, \dots, \vec{\boldsymbol{f}}_c \in G$ with

$$\kappa \in T(\mathfrak{C}_{\vec{\boldsymbol{p}}\to\vec{\boldsymbol{f}}_1})T(\mathfrak{C}_{\vec{\boldsymbol{f}}_1\to\vec{\boldsymbol{f}}_2})\dots T(\mathfrak{C}_{\vec{\boldsymbol{f}}_{c-1}\to\vec{\boldsymbol{f}}_c})T(\mathfrak{C}_{\vec{\boldsymbol{f}}_c\to\vec{\boldsymbol{q}}}) \subseteq T(\mathfrak{C}_{\vec{\boldsymbol{p}}\to\vec{\boldsymbol{q}}})$$

implying $\kappa \in T(\mathfrak{C}_{\vec{\boldsymbol{p}}\to\vec{\boldsymbol{q}}}) \cap \vartheta^{-1}(\#^c) = [\![\gamma]\!]_{\mathfrak{P}}$. ◂

Now, we are able to prove Theorem 10.4.8. To this end, we consider the reachable accepting configurations of $\mathfrak{P}$. Since $\mathfrak{P}$ is a pushdown automaton, this set of configurations is efficiently regular. Now, let $f \in F_{\mathfrak{P}}$ be any final state of $\mathfrak{P}$. Then application of Theorem 9.2.1 yields the effective (and even efficient) regularity of the language of stack contents in configurations $(f, \#^c)$ reachable from $\mathrm{Init}_{\mathfrak{P}}$. From this language we obtain a semi-linear $\mathbb{N}$-language

$$N_f := \{c \in \mathbb{N} \mid (f, \#^c) \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}\} \subseteq \mathbb{N}\,.$$

Finally, we can replace the second factor of the product in Lemma 10.4.13 by W^{N_f-1} (where $N_f - 1 := \{n-1 \mid n \in N_f \setminus \{0\}\} = {}_1\backslash N_f$), which is efficiently recognizable. We will see then that $\textsc{Reach}(L, (W\overline{R})^*)$ is a finite union of such products and, hence, is efficiently recognizable in $\mathbb{M}(\mathcal{A})$.

Proof (of Theorem 10.4.8). Using Theorem 9.2.1 we can compute from $\mathfrak{P}$ and the finite (and, therefore, regular) set $\mathrm{Init}_{\mathfrak{P}}$ of configurations a $\mathfrak{P}$-NFA $\mathfrak{A} = (Q_{\mathfrak{A}}, \{\#\}, Q_{\mathfrak{P}}, \Delta_{\mathfrak{A}}, F_{\mathfrak{A}})$ accepting $C(\mathfrak{A}) = \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}})$. Since $\mathrm{Final}_{\mathfrak{P}}$ also is a regular set of configurations, the set

$$\mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}} = \bigcup_{f\in F_{\mathfrak{P}}} \{f\} \times L(\mathfrak{A}_{f\to F_{\mathfrak{A}}})$$

is regular as well. Note that for any state $f \in F_{\mathfrak{P}}$ the language $L(\mathfrak{A}_{f\to F_{\mathfrak{A}}})$ is semi-linear. In other words, the set $N_f := \{c \in \mathbb{N} \mid \#^c \in L(\mathfrak{A}_{f\to F_{\mathfrak{A}}})\}$ is effectively recognizable in $\mathbb{N}$.

Now, for $f = (s, \vec{\boldsymbol{p}}, \vec{\boldsymbol{q}}) \in F_{\mathfrak{P}}$ we want to compute an asynchronous automaton $\mathfrak{M}_f$ accepting

$$T(\mathfrak{M}_f) = \bigcup_{\#^c\in L(\mathfrak{A}_{f\to F_{\mathfrak{A}}})} \pi_A([\![(f, \#^c)]\!]_{\mathfrak{P}}) = \bigcup_{c\in N_f} \pi_A([\![(f, \#^c)]\!]_{\mathfrak{P}})\,.$$

In other words, this automaton accepts exactly those traces having an abstraction $(f, \#^c)$ with $c \in N_f$, i.e., with $(f, \#^c) \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}$.

To this end, we need the following effectively regular language:

$$Z_f := \begin{cases} T(\mathfrak{C}_{\vec{\boldsymbol{p}}\to\vec{\boldsymbol{q}}}) & \text{if } \varepsilon \in L(\mathfrak{A}_{f\to F_{\mathfrak{A}}}) \text{ (resp. } 0 \in N_f) \\ \varnothing & \text{otherwise}\,. \end{cases}$$

Then we have the following equations due to Lemma 10.4.13:

$$\begin{aligned} T(\mathfrak{M}_f) &= \bigcup_{c \in N_f} \pi_A([\![(f, \#^c)]\!]_{\mathfrak{P}}) \\ &= Z_f \cup \bigcup_{c \in N_f \setminus \{0\}} \big(T(\mathfrak{C}_{\vec{p} \to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A})\big) \cdot W^{c-1} \cdot {}_B\backslash\big(T(\mathfrak{C}_{G \to \vec{q}}) \cap W'\big) \\ &= Z_f \cup \Big(\big(T(\mathfrak{C}_{\vec{p} \to F_{\mathfrak{C}}}) \cap \mathbb{M}(\mathcal{A})\big) \cdot W^{1 \setminus N_f} \cdot {}_B\backslash\big(T(\mathfrak{C}_{G \to \vec{q}}) \cap W'\big)\Big) \end{aligned}$$

where $G = \{\vec{r} \in Q_{\mathfrak{C}} \mid \exists \lambda \in LW'^*\colon I_{\mathfrak{C}} \xrightarrow{\lambda}_{\mathfrak{C}} \vec{r}\}$. This trace language is effectively recognizable in $\mathbb{M}(\mathcal{A})$ by the closure properties of recognizable trace languages (cf. Appendix A) and can be computed in polynomial time for a fixed distributed alphabet $\mathcal{A}$. Finally, we obtain the following equation:

$$\begin{aligned} \textsc{Reach}(L, (W\overline{R})^*) &= \bigcup_{\gamma \in \mathrm{post}^*_{\mathfrak{P}}(\mathrm{Init}_{\mathfrak{P}}) \cap \mathrm{Final}_{\mathfrak{P}}} \pi_A([\![\gamma]\!]_{\mathfrak{P}}) \qquad \text{(by Proposition 10.4.10)} \\ &= \bigcup_{f \in F_{\mathfrak{P}}} \bigcup_{c \in N_f} \pi_A([\![(f, \#^c)]\!]_{\mathfrak{P}}) \\ &= \bigcup_{f \in F_{\mathfrak{P}}} T(\mathfrak{M}_f)\,. \end{aligned}$$

Since there are only polynomial many states in $F_{\mathfrak{P}}$, $\textsc{Reach}(L, (W\overline{R})^*)$ is a union of polynomial many recognizable trace languages. Hence, $\textsc{Reach}(L, (W\overline{R})^*)$ is efficiently recognizable in $\mathbb{M}(\mathcal{A})$. ◂

Further Results

In Theorem 10.4.8 we have only considered forwards reachability in special alternating distributed queue systems. Since the restrictions on read and write actions do not coincide in these meta-transformations we have to consider backwards reachability separately. In this case, utilization of Theorem 10.4.2 yields that $\textsc{BackReach}(L, (W\overline{R})^*)$ is efficiently recognizable whenever $L, W, R \subseteq \mathbb{M}(\mathcal{A})$ are recognizable trace languages such that R is connected. However, we can also consider backwards reachability in alternating distributed queue systems with exactly the restrictions from Theorem 10.4.8. This is considered in the following corollary:

Corollary 10.4.14. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $L, W, R \subseteq \mathbb{M}(\mathcal{A})$ be recognizable such that W is connected. Then the membership problem of $\textsc{BackReach}(L, (W\overline{R})^*)$ is effectively decidable. For a fixed distributed alphabet $\mathcal{A}$ this membership problem is in* P*. However, there are recognizable trace languages $L, W, R \subseteq \mathbb{M}(\mathcal{A})$ where W is connected such that $\textsc{BackReach}(L, (W\overline{R})^*)$ is not even rational.*

Proof. Let $\tau \in \mathbb{M}(\mathcal{A})$ be a trace. To decide $\tau \in \textsc{BackReach}(L, (W\overline{R})^*)$ we first compute an asynchronous automaton $\mathfrak{A}_\tau$ accepting $\textsc{Reach}(\{\tau\}, (W\overline{R})^*)$ as described in Theorem 10.4.8. Finally, we can check in polynomial time whether $T(\mathfrak{A}_\tau) \cap L$ is not empty.

Towards the second statement, recall that

$$\textsc{BackReach}(L, (W\overline{R})^*) = \textsc{Reach}(L^{\mathrm{R}}, (R^{\mathrm{R}}\overline{W^{\mathrm{R}}})^*)^{\mathrm{R}}$$

holds by Theorem 10.4.2 and by $\mathrm{d}((W\overline{R})^*) = (R^{\mathrm{R}}\overline{W^{\mathrm{R}}})^*$. Hence, we have to show that for some recognizable trace languages $L, W, R \subseteq \mathbb{M}(\mathcal{A})$ where R is connected that the language $\textsc{Reach}(L, (W\overline{R})^*)$ is not rational.

Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet with the dependency graph $a - b - c$. Additionally, we set $L := \{\varepsilon\}$, $W := \{acc, b\}$, and $R := \{c\}$ (note that W is not connected). Then we have

$$\textsc{Reach}(L, (W\overline{R})^*) \cap \zeta_{\mathcal{A}}(a^*b^*) = \{[a^n b^n] \mid n \in \mathbb{N}\}.$$

Since there is no regular language $K \subseteq A^*$ satisfying $\zeta_{\mathcal{A}}(K) = \{[a^n b^n] \mid n \in \mathbb{N}\}$, this trace language is not rational (recall that a trace language is rational if it is the image of a regular language since $\zeta_{\mathcal{A}}$ is an epimorphism). Due to the recognizability of the trace language $\zeta_{\mathcal{A}}(a^*b^*)$ and due to the fact that the intersection of a rational trace language with a recognizable one always is rational again, the language $\textsc{Reach}(L, (W\overline{R})^*)$ is not rational. ◂

The first statement of Corollary 10.4.14 also is true if we replace the language of queue contents $L \subseteq \mathbb{M}(\mathcal{A})$ by a rational trace language. This is true, since $\textsc{Reach}(\{\tau\}, (W\overline{R})^*)$ is effectively recognizable (where $\tau \in \mathbb{M}(\mathcal{A})$) and L is rational and, therefore, the intersection of these two trace languages also is effectively rational. Finally, we can check in polynomial time whether this intersection is not empty.

Interestingly, we cannot translate this extension to forwards reachability as the following lemma shows:

Lemma 10.4.15. *There are a distributed alphabet $\mathcal{A} = (A, P, \mathrm{M})$, a rational trace language $L \subseteq \mathbb{M}(\mathcal{A})$, and recognizable trace languages $W, R \subseteq \mathbb{M}(\mathcal{A})$ where W is connected such that $\textsc{Reach}(L, (W\overline{R})^*)$ is an undecidable trace language.*

Proof. We recall Remark 10.4.4. In this lemma, we have given two rational trace languages $L, T \subseteq \mathbb{M}(\mathcal{A})$ such that $\textsc{Reach}(L, \overline{T})$ is undecidable. We only have to adjust our definitions of L, W, and R. Concretely, we set L as given in this remark, $W := \{\varepsilon\}$, and $R := \{[y_j\, cd^j] \mid 1 \leq j \leq k\}$. Then we have $\overline{T} = (W\overline{R})^*$ and, hence, $\varepsilon \in \textsc{Reach}(L, (W\overline{R})^*) = \textsc{Reach}(L, \overline{T})$ if, and only if, the underlying instance of the PCP has a solution. ◂

In other words, we cannot extend Theorem 10.4.8 to trace languages $L \subseteq \mathbb{M}(\mathcal{A})$ of initial queue contents which are not recognizable. The following lemma considers another restriction to Theorem 10.4.8. Concretely, we will see that we really require the language of write traces $W \subseteq \mathbb{M}(\mathcal{A})$ to be connected:

Lemma 10.4.16. *There are a distributed alphabet $\mathcal{A} = (A, P, \mathrm{M})$ and recognizable trace languages $L, W, R \subseteq \mathbb{M}(\mathcal{A})$ where W is not connected such that $\textsc{Reach}(L, (W\overline{R})^*)$ is an undecidable trace language.*

Proof. Let $B = \{a, b\}$ be a binary alphabet and $I = (x_j, y_j)_{1 \le j \le k}$ be an instance of the PCP over the alphabet B. Again, we want to construct L, W, and R with the given restrictions such that $\varepsilon \in \textsc{Reach}(L, (W\overline{R})^*)$ holds if, and only if, I has a solution. To this end, let $\mathcal{A} = (A, P, \mathrm{M})$ be the distributed alphabet from Remark 10.4.4. Then we set

$$W := \{[x_j c d^j] \mid 1 \le j \le k\} \cup \{\varepsilon\}, \quad R := \{[y_j c d^j] \mid 1 \le j \le k\} \cup \{\varepsilon\},$$

and $L := W \setminus \{\varepsilon\}$. Then L, W, and R are finite and, therefore, recognizable. We finally obtain the aforementioned equivalence. ◂

In Section 10.2 we have conjectured that $\textsc{Reach}(L, T^*)$ is regular if L and T are regular and for each two words $s, t \in T$ there is $r \in T$ with $\mathrm{wrt}(r) = \mathrm{wrt}(s)$ and $\mathrm{rd}(r) = \mathrm{rd}(t)$. The lemma from above has shown that this conjecture is not true for distributed queue systems having at least two processes. However, we may conjecture another related problem:

Conjecture 10.4.17. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $L, W_1, \ldots, W_k, R \subseteq \mathbb{M}(\mathcal{A})$ are recognizable where $W_1, \ldots, W_k$ are connected and one of the following properties holds:*

(1) $W_i \parallel W_j$ for each $1 \le i < j \le k$ or

(2) W_i is mono-alphabetic *for each $1 \le i \le k$, i.e., there is $C_i \subseteq A$ such that for each $\sigma \in W_i$ we have $\mathrm{Alph}(\sigma) = C_i$.*

Then $\textsc{Reach}(L, (W_1 W_2 \ldots W_k \overline{R})^*)$ is decidable.

A possible idea to proof this conjecture may be a reduction to the reachability problem of k-counter automata similar to our construction in Theorem 10.4.8. This problem is decidable by [May84] but its complexity is not primitive recursive for unbounded numbers k [CO22, Ler22] and is primitive recursive for fixed k [LS19].

10.4.4 Single Loops

Finally, we want to consider loops of single action traces. Concretely, we want to compute the language $\textsc{Reach}(L, \tau^*)$ for a recognizable trace language $L \subseteq \mathbb{M}(\mathcal{A})$ and an action trace $\tau \in \mathbb{M}(\mathcal{E})$. To compute this trace language we want to utilize our main result from the previous subsection (Theorem 10.4.8). To this end, we should recall that - according to Corollary 4.8.11 - there are three traces $\sigma_1, \sigma_2, \sigma_3 \in \mathbb{M}(\mathcal{A})$ satisfying $\tau \equiv \overline{\sigma_1}\sigma_2\overline{\sigma_3}$. This implies $\textsc{Reach}(L, \tau^*) = \textsc{Reach}(L, (\overline{\sigma_1}\sigma_2\overline{\sigma_3})^*)$. However, until now this does not match the form $\textsc{Reach}(L, (W\overline{R})^*)$ from Theorem 10.4.8. But we know the following equation:

$$(\overline{\sigma_1}\sigma_2\overline{\sigma_3})^* = \{\varepsilon\} \cup \overline{\sigma_1}(\sigma_2\overline{\sigma_3\sigma_1})^*\sigma_2\overline{\sigma_3}.$$

Hence, we can compute $\textsc{Reach}(L, \tau^*)$ by application of Proposition 10.4.3 and computation of $\textsc{Reach}(K, (\sigma_2\overline{\sigma_3\sigma_1})^*)$ for an appropriate trace language $K \subseteq \mathbb{M}(\mathcal{A})$. Therefore, it suffices from now on to consider the case $\textsc{Reach}(L, (\sigma\overline{\rho})^*)$ where $L \subseteq \mathbb{M}(\mathcal{A})$ is recognizable and $\sigma, \rho \in \mathbb{M}(\mathcal{A})$.

Unfortunately, $\textsc{Reach}(L, (\sigma\overline{\rho})^*)$ is not recognizable for any recognizable trace language $L \subseteq \mathbb{M}(\mathcal{A})$ and traces $\sigma, \rho \in \mathbb{M}(\mathcal{A})$. In this connection, recall that $(ab)^*$ is not recognizable if $a \parallel b$ holds. Hence, the map $L \mapsto \textsc{Reach}(L, (ab)^*)$ does not preserve recognizability in this case.

For distributed queue systems having a finite number of independent queues (i.e., the dependence graph $\mathcal{G}_{\mathcal{A}}$ is the disjoint union of some cliques), Boigelot et al. specified in [Boi+97] in which cases the map $L \mapsto \textsc{Reach}(L, \tau^*)$ preserves recognizability. In the context of this paper the authors introduced the notion of a "counter" in an action trace $\tau \in \mathbb{M}(\mathcal{E})$. Such a counter corresponds to a queue which actually writes letters in τ. Then for certain queue inputs we can observe a queue content of the form w^j after j iterations of τ (i.e., we count the number of iterations in this case).

Here, we want to generalize this result to arbitrary distributed alphabets $\mathcal{A}$. To this end, we first need a generalization of counters in an action trace:

Definition 10.4.18. Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\sigma, \rho \in \mathbb{M}(\mathcal{A})$. A subset of letters $C \subseteq A$ is a *counter* in $\sigma\overline{\rho}$ if C is a connected component of the dependence graph $\mathcal{G}_{\mathcal{A}\restriction_{\mathrm{Alph}(\sigma)}}$[xxiii] and one of the following conditions holds:

(1) C consists of one isolated vertex in $\mathcal{A}$ and $|\sigma|_C > |\overline{\rho}|_{\overline{C}}$ or

(2) C contains no isolated vertex in $\mathcal{A}$ and $|\sigma|_C > 0$. ⌟

Before we characterize the action traces $\sigma\overline{\rho}$ where $L \mapsto \textsc{Reach}(L, (\sigma\overline{\rho})^*)$ preserves recognizability, we want to give an intuition on the notion of a counter. So, let $\sigma, \rho \in \mathbb{M}(\mathcal{A})$ be two traces and $C \subseteq A$ be a counter in $\sigma\overline{\rho}$. First we assume that $C \subseteq \mathrm{Isolated}(\mathcal{A})$ consists of one isolated vertex. Then there are $a \in A$ and $i \in P$ with $C = \{a\}$ and $a\,\mathrm{M}\,i$. Since C is a counter there are two numbers $m > n \in \mathbb{N}$ with $\pi_i(\sigma\overline{\rho}) = a^m\overline{a}^n$. According to Lemma 4.8.4(3) we have $a\overline{a} \equiv a$ implying that $a^m\overline{a}^n \equiv a^{m-n} \neq \varepsilon$. Then j iterations of $[\![\sigma\overline{\rho}]\!]$ to an eligible queue content λ results in the content $a^{j\cdot(m-n)}$ in queue i. Hence, queue i counts the iterations.

Now, let $C \cap \mathrm{Isolated}(\mathcal{A}) = \varnothing$ contain no isolated vertex. Then there is $a \in C$ with $\pi_C(\sigma) \in a\mathbb{M}(\mathcal{A})$, another letter $b \in A \setminus \{a\}$ with $a \nparallel b$, and a process $i \in a\,\mathrm{M} \cap b\,\mathrm{M}$. Again, j iterations of $[\![\sigma\overline{\rho}]\!]$ to an eligible queue content λ results in the content $b \cdot \pi_i(\sigma)^j$ in queue i. Therefore, queue i also counts the number of iterations, i.e., the name "counter" is reasonable.

Example 10.4.19. Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet with the dependence graph $a - b - c \quad d$. Then the trace $[acd\overline{a}]$ has the counters $\{a\}$ (note that a is not isolated in $\mathcal{A}$), $\{c\}$, and $\{d\}$. The trace $[acd\overline{d}]$ has the counters $\{a\}$ and $\{c\}$. Moreover, the trace $[abcd\overline{d}]$ only has one counter $\{a, b, c\}$. ⌟

Finally, we can prove the following characterization of the traces $\sigma, \rho \in \mathbb{M}(\mathcal{A})$ preserving the recognizability in $L \mapsto \textsc{Reach}(L, (\sigma\overline{\rho})^*)$:

Theorem 10.4.20. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\sigma, \rho \in \mathbb{M}(\mathcal{A})$. Then the following statements are equivalent:*

(1) For each recognizable trace language $L \subseteq \mathbb{M}(\mathcal{A})$ the set $\textsc{Reach}(L, (\sigma\overline{\rho})^*)$ is (effectively) recognizable.

(2) $\sigma\overline{\rho}$ contains at most one counter.

[xxiii] Note that this is the dependence graph of $\mathcal{A}$ restricted to the write actions occurring in $\sigma\overline{\rho}$.

Proof. First, we assume that $\sigma\overline{\rho}$ contains at most one counter. Our aim is to apply Theorem 10.4.8 to prove the recognizability of $\textsc{Reach}(L, (\sigma\overline{\rho})^*)$. However, until now this is impossible since σ is not necessarily connected since there may be some write actions of an isolated vertex which are superseded by read actions of the same letter. So, let $a \in \text{Isolated}(\mathcal{A})$ be an isolated vertex in the dependence graph of $\mathcal{A}$ with $|\sigma|_a \geq 1$, which is not a counter in $\sigma\overline{\rho}$. Since a is not a counter we have $\pi_i(\sigma\overline{\rho}) = a^m\overline{a}^n$ for two numbers $m \leq n \in \mathbb{N}$. However, from Lemma 4.8.4(3) we know $a\overline{a} \equiv \varepsilon$. Hence, we can compute two traces $\sigma', \rho' \in \mathbb{M}(\mathcal{A})$ with $\sigma\overline{\rho} \equiv \sigma'\overline{\rho'}$, $|\sigma'|_a = 0$, and $|\rho'|_a = n - m$. Finally, iteration of this step yields two traces $\sigma'', \rho'' \in \mathbb{M}(\mathcal{A})$ with $\sigma\overline{\rho} \equiv \sigma''\overline{\rho''}$ and σ'' is connected. Then from Theorem 10.4.8 we obtain the recognizability of $\textsc{Reach}(L, (\sigma\overline{\rho})^*) = \textsc{Reach}(L, (\sigma''\overline{\rho''})^*)$ for any recognizable trace language $L \subseteq \mathbb{M}(\mathcal{A})$.

For the converse implication, let $\sigma\overline{\rho}$ contain at least two counters $C_1, C_2 \subseteq A$. By the definition of a counter we have $C_1 \parallel C_2$. Let $B_1, \ldots, B_k \subseteq A$ be the connected components of $\mathcal{G}_{\mathcal{A}\restriction_{\mathrm{Alph}(\rho)}}$[xxiv]. We can assume that the B_i's are pairwise disjoint. We set $\rho_i := \pi_{B_i}(\rho)$ for each $1 \leq i \leq k$. Since B_i is connected, the trace ρ_i also is connected. We have to distinguish the following cases:

(1) C_1 and C_2 consist of isolated vertices in $\mathcal{A}$. Then there are $a_1, a_2 \in A$ with $C_i = \{a_i\}$. Since $a_i\overline{a_i} \equiv \varepsilon$ holds by Lemma 4.8.4(3), we can assume that $|\rho|_{C_i} > 0$ holds. W.l.o.g. we can assume $C_i = B_i$ (for $i = 1, 2$).

Due to the connectivity of the ρ_i's, the trace language $L := \rho_3^*\rho_4^* \ldots \rho_k^*$ is recognizable in $\mathbb{M}(\mathcal{A})$. Then we have

$$\pi_{\{a_1,a_2\}}(\zeta_{\mathcal{A}}^{-1}(\textsc{Reach}(L, (\sigma\overline{\rho})^*))) \cap a_1^*a_2^* = \left\{a_1^{d_1\cdot n}a_2^{d_2\cdot n} \;\middle|\; n \in \mathbb{N}\right\}$$

where $d_i := |\sigma|_{a_i} - |\rho|_{a_i} \geq 1$ for $i = 1, 2$. This language is not regular. Hence, the word language $\zeta_{\mathcal{A}}^{-1}(\textsc{Reach}(L, (\sigma\overline{\rho})^*))$ is not regular and $\textsc{Reach}(L, (\sigma\overline{\rho})^*)$ is not recognizable in $\mathbb{M}(\mathcal{A})$.

(2) C_1 and C_2 are no isolated vertices in $\mathcal{A}$. For $i = 1, 2$ we set $a_i \in A \cup \{\varepsilon\}$ as follows: if $C_i \cap B_j = \varnothing$ for any $1 \leq j \leq k$ (in this case we do not read any letter from C_i in $\overline{\rho}$) we set $a_i = \varepsilon$. Otherwise we can assume that $C_i \cap B_i \neq \varnothing$ holds. Since the connected component in $\mathcal{G}_{\mathcal{A}}$ containing C_i and B_i has at least two letters, there are two disjoint letters $a_i, b_i \in A$ with $\rho_i \in b_i\mathbb{M}(\mathcal{A})$, $a_i \in C_i$, and $a_i \nparallel b_i$. In this case we observe that $\rho_i \notin a_i\mathbb{M}(\mathcal{A})$.

Independently of $a_i = \varepsilon$ or $a_i \in A$, we set $L := \rho_1^*\rho_2^* \ldots \rho_k^*a_1a_2$. This is a recognizable trace language due to the closure properties of recognizable trace languages. Then we observe

$$\pi_{C_1\cup C_2}(\zeta_{\mathcal{A}}^{-1}(\textsc{Reach}(L, (\sigma\overline{\rho})^*))) \cap a_1a_2w_1^*w_2^* = \{a_1a_2w_1^nw_2^n \mid n \in \mathbb{N}\}$$

where $w_i := \pi_{C_i}(\sigma)$ for $i = 1, 2$. Again, this word language is not regular and, therefore, $\textsc{Reach}(L, (\sigma\overline{\rho})^*)$ is not recognizable.

(3) C_1 is isolated and C_2 is not isolated in $\mathcal{A}$. This is a simple combination of the two cases from above. ◂

One may also check whether the characterization in this theorem generalizes to languages of action traces. However, at least the implication "(1)⇒(2)" is not valid for all pairs $W, R \subseteq$

[xxiv] This is the dependence graph of $\mathcal{A}$ restricted to the read actions occurring in $\sigma\overline{\rho}$.

$\mathbb{M}(\mathcal{A})$ of recognizable trace languages. For example, let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $a, b \in A$ with $a \parallel b$. Moreover, we consider $W = R := \mathbb{M}(\mathcal{A})$. Then $\textsc{Reach}(L, (W\overline{R})^*) = \mathbb{M}(\mathcal{A})$ holds for any trace language $L \subseteq \mathbb{M}(\mathcal{A})$. In this case, (1) is satisfied, but there are traces in $W\overline{R}$ having more than one counter (e.g., $[ab] \in W\overline{R}$).

Until now it is unclear, whether the converse implication is true for any pair $W, R \subseteq \mathbb{M}(\mathcal{A})$ of recognizable trace languages. We conjecture that the language $\textsc{Reach}(L, (W\overline{R})^*)$ is (effectively) recognizable for any recognizable $L \subseteq \mathbb{M}(\mathcal{A})$ if any trace $\sigma\overline{\rho} \in W\overline{R}$ contains at most one counter. Possibly this could be proven by some modifications of our construction in Theorem 10.4.8 like bounded counters for isolated vertices which are "counting down"[xxv].

10.5 Conclusion

In this chapter we have considered the reachability problem of reliable queue automata having one or more queues as well as the reachability problem of partially lossy queue automata. Since this problem is either undecidable or inefficient for all of the mentioned automata models, we have considered an under-approximation which was first introduced by Boigelot et al. in [Boi+97]. So, we approximated the set of reachable queue contents step-by-step where each step consists of a so-called *meta-transformation*. This is a language of action sequences from which we can easily compute the set of reachable queue contents. While Boigelot et al. as well as Abdulla et al. in [Abd+04] considered only single sequences of actions and loops of action sequences, we have considered some further kinds of meta-transformations:

(1) We considered regular (or recognizable) languages which are closed under behavioral equivalence, i.e., under a set of (context-sensitive) commutations of the basic actions.

(2) We also studied multiple loops. Concretely, we considered automata alternating between write and read sequences from two specified regular languages.

In both cases we could prove that, starting from a regular language of queue contents, the queue automaton reaches another regular language of new contents. The constructions proving these results all are possible in polynomial time.

In the light of these kinds of meta-transformations we may also consider a generalization of the so-called *flat queue automata* to automata having a graph consisting of simple paths and single loops as well as components closed under behavioral equivalence or alternating between write and read sequences. Possibly one can prove the decidability (or even NP-completeness) of the reachability problem in such "semi-flat" queue automata similar to the NP-completeness of reachability in flat (lossy) queue automata [FP20, Sch20].

[xxv] An isolated vertex $a \in A$ is a *countdown* in $\sigma\overline{\rho}$ if it occurs in σ and satisfies $|\sigma|_C \leq |\overline{\rho}|_{\overline{C}}$. Then for a queue content λ the iterated application of $\sigma\overline{\rho}$ reduces the number of a's in λ (i.e., the distributed queue system counts down).

Chapter 11
Conclusion

We have investigated the behavior of multiple data types and of automata having a storage mechanism of one of these types. Our studies focused on partially lossy and distributed variations of queues and stacks. Partially lossy queues and stacks are similarly defined as classical queues and stacks. However, these data types are allowed to forget some specified parts of their contents at any time. So, we can see this as a unification of lossy and reliable data types. Automata having a distributed queue or stack as their memory can be seen as a network of finitely many computers each having a queue or stack, respectively, with a special synchronization mechanism between them.

In the first part of this thesis we have considered the transformation monoid of those data types. We have seen that for any sequences of basic actions there is another sequence which is somewhat simple and has the same effect on any content - we call these sequences *normal forms*. This fact turned out to be a helpful tool when verifying automata with those data types. We have also considered some further algebraic properties of the transformation monoid. For example, we have seen that Green's relations are trivial for partially lossy queues but not for partially lossy stacks. In this case there is a strong connection of Green's relations to the aforementioned normal forms.

We have also considered several classes of languages in the transformation monoid of partially lossy queues and stacks. So, the membership problem of rational languages in the transformation monoid of partially lossy queues is NL-complete, while other important problems like equivalence or intersection emptiness inherit their undecidability from Post's correspondence problem [Pos46]. In contrast, we have seen the decidability of almost all of these considered decision problems for partially lossy stacks. For this case we have proven a generalization of a result from Render and Kambites [RK09]. Concretely, we have learned that a language in the transformation monoid of a partially lossy stack is rational if, and only if, the set of its normal forms is regular. With this result we learn that the membership problem and intersection emptiness problem are in P while equivalence is PSPACE-complete.

Another important class we have considered is the class of recognizable languages in the transformation monoid. For partially lossy queues we have proven two characterizations in the style of Kleene's and Büchi's Theorem [Kle51, Büc60]. In other words, we have introduced special rational expressions and a monadic second-order logic describing the recognizable languages. In contrast, for the most partially lossy stacks there are only trivial recognizable languages in their transformation monoid. Only, for counters we have seen that their recognizable languages have a strong connection to finite cyclic groups.

In the second part of this thesis we have studied the reachability problem and the model checking problem of several temporal logics, like LTL and CTL, for automata with partially lossy and distributed stacks and queues. These problems are well-known to be decidable for classical pushdown automata [BEM97, FWW97]. Automata with the other considered data

types are as powerful as Turing-machines. Hence, we have considered special restrictions on these automata.

First, we considered distributed pushdown automata consisting of a local pushdown automaton for each process and in which each transition writes letters only into those local stacks from which we have read a letter right before. For those automata the reachability problem is still decidable in polynomial time - the proof of this decidability is inspired by the constructions from [BEM97, FWW97]. Also recurrent reachability is decidable in polynomial time implying the decidability of the model checking problem of LTL with local properties.

For partially lossy and distributed queue automata we have chosen another restriction. Here, we have extended the under-approximation of the reachability problem which was introduced by Boigelot et al. in [Boi+97]. In this paper the authors approximated the reachability problem step-by-step by merging loops to so-called *meta-transformations*. Concretely, we have studied queue automata applying exactly the action sequences from a recognizable language in their transformation monoid. Additionally, we have considered automata alternating between writing letters from a given regular language of write sequences and reading letters from another regular language of read action sequences. In both cases we have seen that the reachability problem is decidable in polynomial time for those restricted queue automata. Using the aforementioned simple action sequences having the same effect to any queue content, we also obtain the result from Boigelot et al. as a corollary.

Appendix A

Efficient Constructions on Asynchronous Automata

In this chapter we recall the closure properties of the class of recognizable trace languages. All of these closure properties are already known (cf. e.g. [Och85, Die90, DR95]). However, these constructions require a deterministic automata model accepting the recognizable trace languages. Due to this requirement, the constructions for the product or the iteration require in some cases at least exponential time. Here, we present some alternative constructions using the (nondeterministic) asynchronous automata (cf. Definition 9.4.1). In the end, our constructions are possible in polynomial time - at least if we fix the underlying distributed alphabet.

Theorem A.1. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\mathfrak{A}$ and $\mathfrak{B}$ be two asynchronous automata. Then from $\mathfrak{A}$ and $\mathfrak{B}$ we can compute in polynomial time asynchronous automata accepting*

(1) $T(\mathfrak{A}) \cup T(\mathfrak{B})$,

(2) $T(\mathfrak{A}) \cap T(\mathfrak{B})$,

(3) $T(\mathfrak{A})^{\mathrm{R}}$, *and*

(4) $T(\mathfrak{A}) \sqcup\!\sqcup T(\mathfrak{B})$.

Proof idea. These are essentially the constructions for arbitrary NFAs. ◂

Theorem A.2. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\mathfrak{A}$ be an NFA, and $\mathfrak{B}$ be an asynchronous automaton. Then the following statements hold:*

(1) We can compute asynchronous automata accepting ${}_{T(\mathfrak{A})}\backslash T(\mathfrak{B})$ and $T(\mathfrak{B})/_{T(\mathfrak{A})}$, resp., from $\mathfrak{A}$ and $\mathfrak{B}$ in polynomial time.

(2) We can compute NFAs accepting ${}_{T(\mathfrak{B})}\backslash T(\mathfrak{A})$ and $T(\mathfrak{A})/_{T(\mathfrak{B})}$, resp., from $\mathfrak{A}$ and $\mathfrak{B}$. If $\mathcal{A}$ is fixed, these constructions are possible in polynomial time.

Proof idea. The first statement of this theorem is essentially the construction for quotients of regular word languages using NFAs.

Now, we consider the second statement. We first prove the efficient rationality of ${}_{T(\mathfrak{B})}\backslash T(\mathfrak{A})$. To this end, let $\mathfrak{A} = (Q^A, \mathcal{A}, I^A, \Delta^A, F^A)$ and $\mathfrak{B} = (\vec{\boldsymbol{Q}}^{\boldsymbol{B}}, \mathcal{A}, I^B, \Delta^B, F^B)$. We construct an NFA $\mathfrak{C}$ (with ε-transitions) which first simulates $\mathfrak{B}$ with the help of ε-transitions and then $\mathfrak{A}$ with normal transitions. Due to the partial commutations according to the distributed alphabet $\mathcal{A}$, these two simulations can also interleave. This means, even if the simulation of $\mathfrak{B}$ is not finished yet, we can start the simulation of $\mathfrak{A}$ on those processes $\mathfrak{B}$ has already finished. To this end, we have to store the current states of $\mathfrak{A}$ and $\mathfrak{B}$ as well as the set of processes on which the simulation of $\mathfrak{B}$ is already finished.

All in all, the described NFA (with ε-transitions) is $\mathfrak{C} = (Q^C, \mathcal{A}, I^C, \Delta^C, F^C)$ where:

- $Q^C := Q^A \times \vec{\boldsymbol{Q}}^{\boldsymbol{B}} \times 2^P$,
- $I^C := I^A \times I^B \times \{\varnothing\}$,
- $F^C := F^A \times F^B \times 2^P$, and
- Δ^C consists of exactly the following transitions:
  - if there are $p, q \in Q^A$, $\vec{\boldsymbol{r}} \in \vec{\boldsymbol{Q}}^{\boldsymbol{B}}$, $a \in A$, and $X \subseteq P$ with $(p, a, q) \in \Delta^A$ then we have $((p, \vec{\boldsymbol{r}}, X), a, (q, \vec{\boldsymbol{r}}, X \cup a\,\mathrm{M})) \in \Delta^C$.
  - if there are $p, q \in Q^A$, $\vec{\boldsymbol{r}}, \vec{\boldsymbol{s}} \in \vec{\boldsymbol{Q}}^{\boldsymbol{B}}$, $a \in A$, and $X \subseteq P$ with $a\,\mathrm{M} \cap X = \varnothing$, $(p, a, q) \in \Delta^A$, and $(\vec{\boldsymbol{r}}, a, \vec{\boldsymbol{s}}) \in \Delta^B$ then we have $((p, \vec{\boldsymbol{r}}, X), \varepsilon, (q, \vec{\boldsymbol{s}}, X)) \in \Delta^C$.

We can see then $T(\mathfrak{C}) = {}_{T(\mathfrak{B})}\backslash T(\mathfrak{A})$.

Finally, we know $T(\mathfrak{A})/_{T(\mathfrak{B})} = \left({}_{T(\mathfrak{B})^R}\backslash T(\mathfrak{A})^R\right)^R$ and, hence, we obtain the closure under right-quotients using left-quotients and reversal, only. ◂

Theorem A.3. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\mathfrak{A}$ and $\mathfrak{B}$ be two asynchronous automata. Then we can compute an asynchronous automaton accepting $T(\mathfrak{A}) \cdot T(\mathfrak{B})$ from $\mathfrak{A}$ and $\mathfrak{B}$ in polynomial time.*

Proof. Let $\mathfrak{A} = (\vec{\boldsymbol{Q}}^{\boldsymbol{A}}, \mathcal{A}, I^A, \Delta^A, F^A)$ and $\mathfrak{B} = (\vec{\boldsymbol{Q}}^{\boldsymbol{B}}, \mathcal{A}, I^B, \Delta^B, F^B)$ with $\vec{\boldsymbol{Q}}_{\boldsymbol{A}} = \prod_{i \in P} Q_i^A$, $\vec{\boldsymbol{Q}}_{\boldsymbol{B}} = \prod_{i \in P} Q_i^B$, and (w.l.o.g.) $Q_i^A \cap Q_i^B = \varnothing$ for each $i \in P$.

The construction of an asynchronous automaton accepting $T(\mathfrak{A}) \cdot T(\mathfrak{B})$ follows the idea of the classical construction for concatenation of two regular languages for each process $i \in P$. However, it is a bit more involved since we have to ensure that the simulation of $\mathfrak{A}$ ends in a common final state of $\mathfrak{A}$ and the simulation of $\mathfrak{B}$ starts in a common initial state of $\mathfrak{B}$. Concretely, we will construct asynchronous automata accepting $T(\mathfrak{A}_{\vec{\boldsymbol{\iota}}^{\boldsymbol{A}} \to \vec{\boldsymbol{f}}^{\boldsymbol{A}}}) \cdot T(\mathfrak{B}_{\vec{\boldsymbol{\iota}}^{\boldsymbol{B}} \to \vec{\boldsymbol{f}}^{\boldsymbol{B}}})$ for all states $\vec{\boldsymbol{\iota}}^{\boldsymbol{A}} \in I^A$, $\vec{\boldsymbol{f}}^{\boldsymbol{A}} \in F^A$, $\vec{\boldsymbol{\iota}}^{\boldsymbol{B}} \in I^B$, and $\vec{\boldsymbol{f}}^{\boldsymbol{B}} \in F^B$. Then the product $T(\mathfrak{A}) \cdot T(\mathfrak{B})$ is the union of all combinations of initial and final states of $\mathfrak{A}$ and $\mathfrak{B}$.

Now, let $\vec{\boldsymbol{\iota}}^{\boldsymbol{A}} \in I^A$, $\vec{\boldsymbol{f}}^{\boldsymbol{A}} \in F^A$, $\vec{\boldsymbol{\iota}}^{\boldsymbol{B}} \in I^B$, and $\vec{\boldsymbol{f}}^{\boldsymbol{B}} \in F^B$. We now construct the following asynchronous automaton $\mathfrak{C} = (\vec{\boldsymbol{Q}}^{\boldsymbol{C}}, \mathcal{A}, I^C, \Delta^C, F^C)$ accepting $T(\mathfrak{A}_{\vec{\boldsymbol{\iota}}^{\boldsymbol{A}} \to \vec{\boldsymbol{f}}^{\boldsymbol{A}}}) \cdot T(\mathfrak{B}_{\vec{\boldsymbol{\iota}}^{\boldsymbol{B}} \to \vec{\boldsymbol{f}}^{\boldsymbol{B}}})$:

- $\vec{\boldsymbol{Q}}^{\boldsymbol{C}} := \prod_{i \in P} Q_i^C$ with $Q_i^C := Q_i^A \cup Q_i^B$,

- $I^C := \left\{ (p_i)_{i\in P} \middle| \forall\, i \in P\colon p_i \in \begin{cases} \{\iota_i^A, \iota_i^B\} & \text{if } \iota_i^A = f_i^A \\ \{\iota_i^A\} & \text{otherwise} \end{cases} \right\}$,
- $F^C := \left\{ (p_i)_{i\in P} \middle| \forall\, i \in P\colon p_i \in \begin{cases} \{f_i^A, f_i^B\} & \text{if } \iota_i^B = f_i^B \\ \{f_i^B\} & \text{otherwise} \end{cases} \right\}$, and
- $((p_i)_{i\in a\,\mathrm{M}}, (q_i)_{i\in a\,\mathrm{M}}) \in \Delta_a^C$ iff for each $i \in a\,\mathrm{M}$ there is $r_i \in Q_i^C$ such that
  (1) $r_i = q_i$ or
  (2) $r_i = f_i^A$ and $q_i = \iota_i^B$
  and $((p_i)_{i\in a\,\mathrm{M}}, (r_i)_{i\in a\,\mathrm{M}}) \in \Delta_a^A \cup \Delta_a^B$.

Finally, we obtain an asynchronous automaton accepting $T(\mathfrak{A}) \cdot T(\mathfrak{B})$ by the union of all these trace languages $T(\mathfrak{C}) = T(\mathfrak{A}_{\vec{\iota}^A \to \vec{f}^A}) \cdot T(\mathfrak{B}_{\vec{\iota}^B \to \vec{f}^B})$. This is possible in polynomial time since there are only polynomial many combinations of initial and final states and due to Theorem A.1. ◂

Theorem A.4. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet and $\mathfrak{A}$ be an asynchronous automaton such that $T(\mathfrak{A})$ is connected. Then we can compute an asynchronous automaton accepting $T(\mathfrak{A})^*$ from $\mathfrak{A}$. If $\mathcal{A}$ is fixed, this construction is possible in polynomial time.*

Proof. The idea of our construction is the following: first, we partition $T(\mathfrak{A})$ into recognizable languages of traces having a "similar" run in $\mathfrak{A}$. Concretely, for each pair $\vec{\boldsymbol{\iota}}$ and $\vec{\boldsymbol{f}}$ of initial and final states in $\mathfrak{A}$ and each connected subset $C \subseteq A$ we construct an asynchronous automaton $\mathfrak{B}$ accepting exactly those traces $\lambda \in \mathbb{M}(\mathcal{A})$ which have a run from $\vec{\boldsymbol{\iota}}$ to $\vec{\boldsymbol{f}}$ in $\mathfrak{A}$ and satisfy $\mathrm{Alph}(\lambda) = C$. Additionally, we attach a label to each letter which can be read by these automata. These labels consist of the set C and a small natural number. They turn out to be very helpful later in our proof. Concretely, we have to distinguish in our proof to which subtrace a letter belongs to in a sequence of multiple traces from $T(\mathfrak{A})$. Then such label is a very important indicator for this mapping. After the construction of the described automata accepting traces with similar runs, we utilize the classical construction for the Kleene-star on the disjoint union of the aforementioned automata. Finally, we are able to remove the labels and obtain the Kleene-closure of $T(\mathfrak{A})$.

First, we consider the case $\varepsilon \in T(\mathfrak{A})$. Then we have $T(\mathfrak{A})^* = (T(\mathfrak{A}) \setminus \{\varepsilon\})^*$. Hence, from now on we can assume $\varepsilon \notin T(\mathfrak{A})$.

Now, we define our labels: a *label* is a tuple $\mathcal{L} = (C, z)$ where $C \subseteq A$ is a non-empty connected set in $\mathcal{A}$ and $0 \leq z \leq 2|P|$ is a (small) natural number. The flag z will help us to distinguish consecutive traces having the same induced alphabet C. By Λ we denote the set of all labels. We blow up our distributed alphabet $\mathcal{A}$ with labels as follows: $\mathcal{A}' = (A', P, \mathrm{M}')$ is the distributed alphabet with

- $A' := \{(a, \mathcal{L}) \mid \mathcal{L} = (C, z) \in \Lambda, a \in C\}$ and
- $\mathrm{M}' := \{((a, \mathcal{L}), i) \mid (a, \mathcal{L}) \in A', a\,\mathrm{M}\,i\}$.

We also define a transduction $\pi\colon A' \to A\colon (a, \mathcal{L}) \mapsto a$. Since π preserves dependencies of letters, the transduction $\pi(L)$ of a recognizable trace language $L \subseteq \mathbb{M}(\mathcal{A}')$ is recognizable as well.

Let $\mathfrak{A} = (\vec{\boldsymbol{Q}}, \mathcal{A}, I, \Delta, F)$. Then $\pi^{-1}(T(\mathfrak{A}))$ is effectively recognizable. Additionally, let $\vec{\boldsymbol{\imath}} \in I$, $\vec{\boldsymbol{f}} \in F$ be a pair of initial and final states of $\mathfrak{A}$ and $\mathcal{L} = (C, z)$ be a label. We can construct an asynchronous automaton $\mathfrak{A}^{\vec{\imath},\vec{f},\mathcal{L}} = (\vec{\boldsymbol{Q}}^{\vec{\imath},\vec{f},\mathcal{L}}, \mathcal{A}', I^{\vec{\imath},\vec{f},\mathcal{L}}, \Delta^{\vec{\imath},\vec{f},\mathcal{L}}, F^{\vec{\imath},\vec{f},\mathcal{L}})$ accepting

$$\pi^{-1}\Big(T(\mathfrak{A}_{\vec{\imath}\to\vec{f}})\Big) \cap \bigcap_{a\in C} \mathbb{M}(\mathcal{A}'\restriction_{C\times\{\mathcal{L}\}}) \cdot (a, \mathcal{L}) \cdot \mathbb{M}(\mathcal{A}'\restriction_{C\times\{\mathcal{L}\}}).$$

In other words, $\mathfrak{A}^{\vec{\imath},\vec{f},\mathcal{L}}$ accepts exactly those $\mathcal{L}$-labeled traces $\lambda \in \mathbb{M}(\mathcal{A}')$ such that $\pi(\lambda)$ has a run from $\vec{\boldsymbol{\imath}}$ to $\vec{\boldsymbol{f}}$ in $\mathfrak{A}$ and $\mathrm{Alph}(\lambda) = C \times \{\mathcal{L}\}$. Note that there is an asynchronous automaton accepting $\mathbb{M}(\mathcal{A}'\restriction_{C\times\{\mathcal{L}\}}) \cdot (a, \mathcal{L}) \cdot \mathbb{M}(\mathcal{A}'\restriction_{C\times\{\mathcal{L}\}})$ where each process $i \in P$ has at most two local states.

We also have to store a bounded part of the history of labels, our asynchronous automaton has seen on each of its processes. These histories will prevent our automaton to read a shuffle of two or more traces which is not serializable anymore. For example, Figure A.1 shows the shuffle of two traces $\kappa, \lambda \in T(\mathfrak{A})$ which is neither equivalent to $\kappa\lambda$ nor to $\lambda\kappa$.

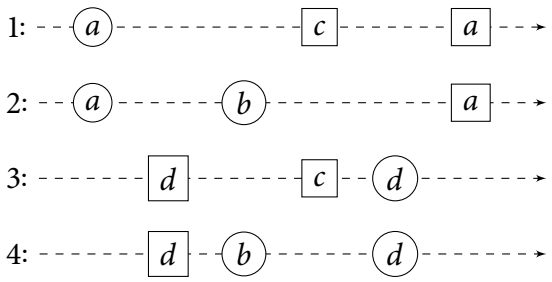

Figure A.1. A shuffle σ of $\kappa = [abd]$ (circles) and $\lambda = [dca]$ (rectangles). On each process $i \in P$ we have $\pi_{A_i}(\sigma) = \pi_{A_i}(\kappa\lambda)$ or $\pi_{A_i}(\sigma) = \pi_{A_i}(\lambda\kappa)$, i.e., κ and λ are sequential on each process. However, σ cannot be serialized into $\kappa\lambda$ or $\lambda\kappa$.

So, we have to implement a mechanism which prevents such situation. To this end, we introduce a so-called *history*. Such history is a special trace on the distributed alphabet $\mathcal{H} = (\Lambda, P, \mathrm{N})$ with $\mathrm{N} = \{((C, z), i) \mid i \in C\,\mathrm{M}\}$. Concretely, a trace $H \in \mathbb{M}(\mathcal{H})$ is a *history* if each label $\mathcal{L} \in \Lambda$ occurs in H at most once. We also have to merge multiple histories: $H_1 \vee H_2$ for two histories H_1 and H_2 is constructed in three steps as follows:

1. **Pruning.** If there is a connected set $C \subseteq A$ occurring more than $|P|$ times in H_1, we remove all of the labels with alphabet C except for the $|P|$ right-most occurrences of C. We also do this operation on H_2.
2. **Joining.** In this step we see H_1 and H_2 as two strict partial orderings on Λ: We have $(C, y) <_j (D, z)$ if $C \nparallel D$ and $H_j \in \mathbb{M}(\mathcal{H}) \cdot (C, y) \cdot \mathbb{M}(\mathcal{H}) \cdot (D, z) \cdot \mathbb{M}(\mathcal{H})$ (for $j = 1, 2$). Finally, by $<$ we denote the transitive closure of $<_1 \cup <_2$.
3. **Checking.** If $<$ is a strict partial ordering, we obtain a joint history H. Otherwise, $<$ is not irreflexive or not anti-symmetric. In this case, $<$ induces no trace and, therefore, our merge operation fails.

Note that we can see the letters of the histories H_1 and H_2 as the (interlocking) teeth of a zipper. Then our merge-operation $\curlyvee$ can be visualized as the closing of this zipper.

▷ *Example.* Consider the situation in Figure A.1. Let $\mathcal{K}$ and $\mathcal{L}$ be labels of the traces κ resp. λ (note $\mathcal{K} \neq \mathcal{L}$). After reading b, the process 4 learns the history $\mathcal{L}\mathcal{K}$. After reading c, the process 3 stores $\mathcal{K}\mathcal{L}$. Finally, when reading the second d we have to merge the histories $\mathcal{L}\mathcal{K}$ and $\mathcal{K}\mathcal{L}$ which is impossible. Hence, our automaton will block in this moment. ⌟

Now, we apply the classical construction for the Kleene-star on the disjoint union of all automata $\mathfrak{A}^{\vec{\imath},\vec{f},\mathcal{L}}$ with some special restrictions. So, $\mathfrak{B} = (\vec{\boldsymbol{Q}}^{\boldsymbol{B}}, \mathcal{A}', I^B, \Delta^B, F^B)$ is the following asynchronous automaton:

- $\vec{\boldsymbol{Q}}^{\boldsymbol{B}} := \prod_{i\in P} Q_i^B$ and $Q_i^B := \left\{ (q, \vec{\imath}, \vec{f}, \mathcal{L}, H, m) \;\middle|\; \begin{array}{l} q \in Q_i^{\vec{\imath},\vec{f},\mathcal{L}}, H \text{ is a history,} \\ m\colon 2^A \to \{0,\dots,2|P|\} \end{array} \right\} \cup \{\top_i\}$,
- $I^B := \{(q_i, \vec{\boldsymbol{\imath}}_{\boldsymbol{i}}, \vec{\boldsymbol{f}}_{\boldsymbol{i}}, \mathcal{L}_i, H_i, \vec{\boldsymbol{0}})_{i\in P} \mid q_i \in I_i^{\vec{\imath}_i,\vec{f}_i,\mathcal{L}_i}, H_i = \mathcal{L}_i = (C_i, 0)\} \cup \{(\top_i)_{i\in P}\}$, where $\vec{\boldsymbol{0}}\colon 2^A \to \{0,\dots,2|P|\}\colon C \mapsto 0$ maps all $C \subseteq A$ to 0,
- $F^B := \{(q_i, \vec{\boldsymbol{\imath}}_{\boldsymbol{i}}, \vec{\boldsymbol{f}}_{\boldsymbol{i}}, \mathcal{L}_i, H_i, m_i)_{i\in P} \mid q_i \in F_i^{\vec{\imath}_i,\vec{f}_i,\mathcal{L}_i}\} \cup \{(\top_i)_{i\in P}\}$, and
- $((p_i, \vec{\imath}, \vec{f}, \mathcal{L}, G_i, m_i)_{i\in a\,\mathrm{M}}, (q_i, \vec{\boldsymbol{f}}_{\boldsymbol{i}}, \vec{\boldsymbol{g}}_{\boldsymbol{i}}, \mathcal{K}_i, H_i, n_i)_{i\in a\,\mathrm{M}}) \in \Delta^B_{(a,\mathcal{L})}$ iff $H := \curlyvee_{i\in a\,\mathrm{M}} G_i$ is well-defined and for each $i \in a\,\mathrm{M}$ there is $r_i \in Q_i^{\vec{\imath},\vec{f},\mathcal{L}}$ such that
  - (i) $r_i = q_i$ and $(\vec{\imath}, \vec{f}, \mathcal{L}, H, m_i) = (\vec{\boldsymbol{f}}_{\boldsymbol{i}}, \vec{\boldsymbol{g}}_{\boldsymbol{i}}, \mathcal{K}_i, H_i, n_i)$ or
  - (ii) $r_i \in F_i^{\vec{\imath},\vec{f},\mathcal{L}}$ and $q_i \in I_i^{\vec{f}_i,\vec{g}_i,\mathcal{K}_i}$, $\mathcal{K}_i = (C, z)$ with $z := (m_i(C) + 1) \bmod (2|P| + 1)$, $H_i := H \cdot \mathcal{K}_i$ (i.e., we start reading a $\mathcal{K}_i$-labeled trace after this transition), and n_i arises from m_i by increasing the value of C modulo $2|P| + 1$

  and $((p_i)_{i\in a\,\mathrm{M}}, (r_i)_{i\in a\,\mathrm{M}}) \in \Delta^{\vec{\imath},\vec{f},\mathcal{L}}_{(a,\mathcal{L})}$.

Then $\mathfrak{B}$ has size $O(|\mathfrak{A}| \cdot |Q|^2 \cdot (2^{|A|})! \cdot 2^{|A|\cdot|P|})$, i.e., if $\mathcal{A}$ is fixed this construction is possible in polynomial time (and in twofold exponential time if $\mathcal{A}$ is not fixed). Now, we have to prove $\pi(T(\mathfrak{B})) = T(\mathfrak{A})^*$.

▷ *Claim 1.* *We have* $T(\mathfrak{A})^* \subseteq \pi(T(\mathfrak{B}))$.

Proof. Let $\lambda \in T(\mathfrak{A})^*$. Then there are traces $\kappa_1, \dots, \kappa_k \in T(\mathfrak{A})$ with $\lambda = \kappa_1 \dots \kappa_k$. If $k = 0$ then we have $\lambda = \varepsilon$. Since $(\top_i)_{i\in P} \in I^B \cap F^B$ we also have $\varepsilon \in \pi(T(\mathfrak{B}))$. So, from now on, assume $k > 0$. Then there are accepting runs in $\mathfrak{A}$ labeled with κ_n for each $1 \leq n \leq k$, i.e., there are $a_{n,1}, \dots, a_{n,\ell_n} \in A$ and $\vec{\boldsymbol{q}}_{\boldsymbol{n},\boldsymbol{0}}, \dots, \vec{\boldsymbol{q}}_{\boldsymbol{n},\boldsymbol{\ell_n}} \in \vec{\boldsymbol{Q}}$ with $\kappa_n = [a_{n,1} \dots a_{n,\ell_n}]$, $\vec{\boldsymbol{q}}_{\boldsymbol{n},\boldsymbol{0}} \in I$, $\vec{\boldsymbol{q}}_{\boldsymbol{n},\boldsymbol{j-1}} \xrightarrow{a_{n,j}}_{\mathfrak{A}} \vec{\boldsymbol{q}}_{\boldsymbol{n},\boldsymbol{j}}$ for each $1 \leq j \leq \ell_n$, and $\vec{\boldsymbol{q}}_{\boldsymbol{n},\boldsymbol{\ell_n}} \in F$ for each $1 \leq n \leq k$.

For $1 \leq n \leq k$ let $\mathcal{L}_n = (C_n, z_n)$ be a label such that the flag z_n is - starting from 0 - increasing monotonically (modulo $2|P| + 1$) for labels with the same connected alphabet $C_n := \mathrm{Alph}(\kappa_n)$.

We now construct a run $\vec{\boldsymbol{p}}_{\boldsymbol{1},\boldsymbol{0}}, \dots, \vec{\boldsymbol{p}}_{\boldsymbol{1},\boldsymbol{\ell_1}}, \dots, \vec{\boldsymbol{p}}_{\boldsymbol{k},\boldsymbol{0}}, \dots, \vec{\boldsymbol{p}}_{\boldsymbol{k},\boldsymbol{\ell_k}} \in \vec{\boldsymbol{Q}}^{\boldsymbol{B}}$ labeled with

$$\lambda' := [(a_{1,1}, \mathcal{L}_1) \dots (a_{1,\ell_1}, \mathcal{L}_1) \dots (a_{k,1}, \mathcal{L}_k) \dots (a_{k,\ell_k}, \mathcal{L}_k)]$$

in $\mathfrak{B}$. We do this by induction on $1 \leq n \leq k$ and $0 \leq j \leq \ell_n$. We start with $n = 1$ and $j = 0$. We set $\vec{\boldsymbol{p}}_{\boldsymbol{1},\boldsymbol{0}} := (p_{x_i,0,i}, \vec{\boldsymbol{q}}_{\boldsymbol{x_i},\boldsymbol{0}}, \vec{\boldsymbol{q}}_{\boldsymbol{x_i},\boldsymbol{\ell_{x_i}}}, \mathcal{L}_{x_i}, H_i, \vec{\boldsymbol{0}})_{i\in P}$ where

- $1 \leq x_i \leq k$ is minimal with $i \in C_{x_i}\,\mathrm{M}$ (or $x_i := 1$ if this minimum does not exist),

- $p_{x_i,0,i}$ is the i^{th} component of $\vec{\boldsymbol{q}}_{x_i,\mathbf{0}}$, and
- $H_i := \mathcal{L}_{x_i}$.

Then, by the choice of the components of $\vec{\boldsymbol{p}}_{\mathbf{1},\mathbf{0}}$ we obtain $\vec{\boldsymbol{p}}_{\mathbf{1},\mathbf{0}} \in I^B$.

Next, let $1 \leq n \leq k$ and $0 \leq j \leq \ell_n$. We set

$$\vec{\boldsymbol{p}}_{n,j} := (p_{n,j,i}, \vec{\boldsymbol{\iota}}_{n,j,i}, \vec{\boldsymbol{f}}_{n,j,i}, \mathcal{L}_{n,j,i}, H_{n,j,i}, m_{n,j,i})_{i \in P}$$

as follows:

(1) if $n > 1$ and $j = 0$ we set $\vec{\boldsymbol{p}}_{n,\mathbf{0}} = \vec{\boldsymbol{p}}_{n-\mathbf{1},\ell_{n-1}}$.

(2) otherwise we have to consider each $i \in P$ separately. If $i \notin a_{n,j}\,\mathrm{M}$, we leave the i^{th} component of $\vec{\boldsymbol{p}}_{n,j-1}$ untouched. Now, assume $i \in a_{n,j}\,\mathrm{M}$. Then we have to distinguish the following cases:

(2.1) $i \in \{a_{n,j+1}, \ldots, a_{n,\ell_n}\}\,\mathrm{M}$ (i.e., while reading the remaining letters of κ_n we will also modify the i^{th} component). Then we set $p_{n,j,i}$ to the i^{th} component of $\vec{\boldsymbol{q}}_{n,j}$, $H_{n,j,i} := \bigvee_{x \in a_{n,j}\,\mathrm{M}} H_{n,j-1,x}$ (note that this is well-defined since we sequentially process the κ_n's), and the remaining components are copied from $\vec{\boldsymbol{p}}_{n,j-1}$. In other words, we apply a transition of type (i).

(2.2) $i \notin \{a_{n,j+1}, \ldots, a_{n,\ell_n}\}\,\mathrm{M}$ and $i \notin \bigcup_{n<x\leq k} C_x\,\mathrm{M}$ (i.e., neither κ_n nor $\kappa_{n+1} \ldots \kappa_k$ contain any letter modifying the i^{th} component). This case is similar to the first case, since we will end up in a final state in the i^{th} component and remain in this state until the end of this computation.

(2.3) $i \notin \{a_{n,j+1}, \ldots, a_{n,\ell_n}\}\,\mathrm{M}$ and $i \in \bigcup_{n<x\leq k} C_x\,\mathrm{M}$ (i.e., the suffix of κ_n does not modify the i^{th} component, but $\kappa_{n+1} \ldots \kappa_k$ will do). Then there is $n < x \leq k$ minimal with $i \in C_x\,\mathrm{M}$. We set:

- $p_{n,j,i}$ is the i^{th} component of $\vec{\boldsymbol{q}}_{x,\mathbf{0}}$,
- $\vec{\boldsymbol{\iota}}_{n,j,i} := \vec{\boldsymbol{q}}_{x,\mathbf{0}}$,
- $\vec{\boldsymbol{f}}_{n,j,i} := \vec{\boldsymbol{q}}_{x,\ell_x}$,
- $\mathcal{L}_{n,j,i} := \mathcal{L}_x$,
- $H_{n,j,i} := \left(\bigvee_{y \in a\,\mathrm{M}} H_{n,j-1,y}\right) \cdot \mathcal{L}_x$, and
- $m_{n,j,i} := m_{n,j-1,i}[C_x \mapsto (z_x + 1) \bmod (2|P| + 1)]$ where $\mathcal{L}_x = (C_x, z_x)$.

In other words, we apply a transition of type (ii).

By our construction we obtain $\vec{\boldsymbol{p}}_{n,j-1} \xrightarrow{(a_{n,j},\mathcal{L}_n)}_{\mathfrak{B}} \vec{\boldsymbol{p}}_{n,j}$ for each $1 \leq n \leq k$ and $0 \leq j \leq \ell_n$ as well as $\vec{\boldsymbol{p}}_{k,\ell_k} \in F^B$. Hence, $\vec{\boldsymbol{p}}_{\mathbf{1},\mathbf{0}}, \ldots, \vec{\boldsymbol{p}}_{\mathbf{1},\boldsymbol{\ell}_1} = \vec{\boldsymbol{p}}_{\mathbf{2},\mathbf{0}}, \ldots, \vec{\boldsymbol{p}}_{\boldsymbol{k},\boldsymbol{\ell}_k}$ is a run in $\mathfrak{B}$ which starts in $\vec{\boldsymbol{p}}_{\mathbf{1},\mathbf{0}} \in I^B$, ends in $\vec{\boldsymbol{p}}_{\boldsymbol{k},\boldsymbol{\ell}_k} \in F^B$, and is labeled with λ'. In other words, we have $\lambda' \in T(\mathfrak{B})$ implying $\lambda = \pi(\lambda') \in \pi(T(\mathfrak{B}))$. ◁

▷ *Claim 2.* *We have* $\pi(T(\mathfrak{B})) \subseteq T(\mathfrak{A})^*$.

Proof. Let $\lambda \in \pi(T(\mathfrak{B}))$. Then there is $\lambda' \in T(\mathfrak{B})$ with $\lambda = \pi(\lambda')$. There are (labeled) letters $(a_1, \mathcal{L}_1), \ldots, (a_\ell, \mathcal{L}_\ell) \in A'$ and states $\vec{\boldsymbol{p}}_{\mathbf{0}}, \ldots, \vec{\boldsymbol{p}}_{\boldsymbol{\ell}} \in \vec{\boldsymbol{Q}}^{\boldsymbol{B}}$ with $\vec{\boldsymbol{p}}_{\mathbf{0}} \in I^B$, $\vec{\boldsymbol{p}}_{\boldsymbol{\ell}} \in F^B$, $\vec{\boldsymbol{p}}_{\boldsymbol{j}-\mathbf{1}} \xrightarrow{(a_j,\mathcal{L}_j)}_{\mathfrak{B}} \vec{\boldsymbol{p}}_{\boldsymbol{j}}$ for each $1 \leq j \leq \ell$, and $\lambda' = [(a_1, \mathcal{L}_1) \ldots (a_\ell, \mathcal{L}_\ell)]$ (i.e., we have $\lambda = [a_1 \ldots a_\ell]$). If $\vec{\boldsymbol{p}}_{\mathbf{0}} = (\top_i)_{i \in P}$, we have $\ell = 0$ and, hence, $\lambda = \varepsilon \in T(\mathfrak{A})^*$. So, from now on we assume $\vec{\boldsymbol{p}}_{\mathbf{0}} \neq (\top_i)_{i \in P}$. Let

$$\vec{\boldsymbol{p}}_{\boldsymbol{j}} = (p_{j,i}, \vec{\boldsymbol{\iota}}_{\boldsymbol{j},\boldsymbol{i}}, \vec{\boldsymbol{f}}_{\boldsymbol{j},\boldsymbol{i}}, \mathcal{L}_{j,i}, H_{j,i}, m_{j,i})_{i \in P}\,.$$

Now, let $1 \leq j \leq \ell$. Then there is a label $\mathcal{L} = (C, z)$ and a maximal sequence $1 \leq n_1 < n_2 < \cdots < n_k \leq \ell$ such that the following properties hold:

(1) $j = n_x$ for a $1 \leq x \leq k$ (i.e., j belongs to this sequence),

(2) $\mathcal{L}_{n_x} = \mathcal{L}$ (i.e., all letters with indexes n_x are labeled with $\mathcal{L}$), and

(3) for each $n_1 < y < n_k$ with $y \neq n_x$ for each $1 \leq x \leq k$ we have $\mathcal{L}_y = (C', z')$ with $C' \neq C$ (i.e., we do not see other labels with alphabet C in the range of this subsequence).

In other words, $n_1, \ldots, n_k$ is the maximal subsequence of "consecutive" (except for partial commutations) indices of letters that are labeled with $\mathcal{L}$. By definition of $\mathfrak{B}$ there are $\vec{\imath} \in I$ and $\vec{f} \in F$ such that $\mathfrak{B}$ simulates a run from $I^{\vec{\imath},\vec{f},\mathcal{L}}$ to $F^{\vec{\imath},\vec{f},\mathcal{L}}$ in $\mathfrak{A}^{\vec{\imath},\vec{f},\mathcal{L}}$ which is labeled by $(a_{n_1}, \mathcal{L}) \ldots (a_{n_k}, \mathcal{L})$. Hence, we have $(a_{n_1}, \mathcal{L}) \ldots (a_{n_k}, \mathcal{L}) \in T(\mathfrak{A}^{\vec{\imath},\vec{f},\mathcal{L}})$ and $a_{n_1} \ldots a_{n_k} \in T(\mathfrak{A})$.

Now, let $\kappa_1, \ldots, \kappa_m \in T(\mathfrak{A})$ be all traces we have constructed in this way. From the histories constructed by the asynchronous automaton $\mathfrak{B}$ (note that all of them are well-defined traces), we obtain a topological ordering τ of $\{1, \ldots, m\}$ such that $\kappa_{\tau(1)} \ldots \kappa_{\tau(m)} = [a_1 \ldots a_\ell] = \lambda$. Hence, we obtain $\lambda \in T(\mathfrak{A})^*$. ◁

Finally, since $\pi(T(\mathfrak{B}))$ is efficiently recognizable (we only replace letters $(a, \mathcal{L})$ by a in $\mathfrak{B}$), we are done. ◂

Corollary A.5. *Let $\mathcal{A} = (A, P, \mathrm{M})$ be a distributed alphabet, $\mathfrak{A}$ be an asynchronous automaton such that $T(\mathfrak{A})$ is connected, and $S \subseteq \mathbb{N}$ be recognizable in $\mathbb{N}$. Then we can compute an asynchronous automaton accepting $T(\mathfrak{A})^S = \bigcup_{n \in S} T(\mathfrak{A})^n$ from $\mathfrak{A}$ and an automaton accepting S. If $\mathcal{A}$ is fixed, this construction is possible in polynomial time.*

Proof idea. We know that $S \subseteq \mathbb{N}$ is recognizable in $\mathbb{N}$ if, and only if, S is semi-linear. Hence, there are constants $k_1, \ldots, k_n, \ell_1, \ldots, \ell_n \in \mathbb{N}$ such that

$$S = \bigcup_{1 \leq i \leq n} k_i \cdot \mathbb{N} + \ell_i \,,$$

i.e., S is the union of finitely many linear progressions. Hence, we obtain the following equation

$$T(\mathfrak{A})^S = \bigcup_{1 \leq i \leq n} (T(\mathfrak{A})^{k_i})^* \cdot T(\mathfrak{A})^{\ell_i} \,.$$

To prove the recognizability of $T(\mathfrak{A})^S$ it suffices to show that $(T(\mathfrak{A})^{k_i})^*$ is recognizable in $\mathbb{M}(\mathcal{A})$. We prove this with the help of an extension of the construction in Theorem A.4 as follows: to each state of the constructed automaton $\mathfrak{B}$ we add yet another component, which is a number $0 \leq c < k_i$. In this component, we want to count (modulo k_i) the number of traces from $T(\mathfrak{A})$ we have seen on a run of $\mathfrak{B}$.

However, it is not a good idea to count this number on each process since (for example) some traces may affect two and others may affect three processes. In this case we would be unable to decide whether to accept at the end of a run, or not.

Hence, we have to determine one of the processes associated to a trace $\lambda \in T(\mathfrak{A})$ which counts its occurrences. We do this with the help of an arbitrary function $\xi \colon 2^A \setminus \{\varnothing\} \to P$ with $\xi(C) \in C\,\mathrm{M}$. Then, whenever we read a trace $\lambda \in T(\mathfrak{A})$, we only increase the counter on the process with number $\xi(\mathrm{Alph}(\lambda))$. Finally, the automaton accepts whenever we are in an accepting state as in $\mathfrak{B}$ (cf. Theorem A.4) and the sum of all counters is zero modulo k_i.

By closure properties we obtain an asynchronous automaton accepting $T(\mathfrak{A})^S$. ◂

Resulting Publications

[AK18] F. Abu Zaid and C. Köcher. *The Cayley-Graph of the Queue Monoid: Logic and Decidability*. In: *FSTTCS 2018*. Vol. 122. LIPIcs. Dagstuhl Publishing, 2018, 9:1–9:17. DOI: 10.4230/LIPIcs.FSTTCS.2018.9.

[KK17] C. Köcher and D. Kuske. *The Transformation Monoid of a Partially Lossy Queue*. In: *CSR 2017*. Vol. 10304. LNCS. Springer, 2017, pp. 191–205. DOI: 10.1007/978-3-319-58747-9_18 (cited on p. 233).

[KK22] C. Köcher and D. Kuske. *Forwards- and Backwards-Reachability for Cooperating Multi-Pushdown Systems*. Submitted. 2022 (cited on p. 138).

[KKP18] C. Köcher, D. Kuske, and O. Prianychnykova. *The Inclusion Structure of Partially Lossy Queue Monoids and Their Trace Submonoids*. In: RAIRO - Theoretical Informatics and Applications 52.1 (2018). Full paper of [KK17], pp. 55–86. DOI: 10.1051/ita/2018003 (cited on pp. 4, 22, 25, 35, 50, 60 sq., 63, 72, 84, 193).

[Köc18] C. Köcher. *Rational, Recognizable, and Aperiodic Sets in the Partially Lossy Queue Monoid*. In: STACS 2018. Vol. 96. LIPIcs. Dagstuhl Publishing, 2018, 45:1–45:14. DOI: 10.4230/LIPIcs.STACS.2018.45 (cited on p. 233).

[Köc19] C. Köcher. *Reachability Problems on Partially Lossy Queue Automata*. In: RP 2019. Vol. 11674. LNCS. Springer, 2019, pp. 149–163. DOI: 10.1007/978-3-030-30806-3_12 (cited on pp. 174, 233).

[Köc21] C. Köcher. *Reachability Problems on Reliable and Lossy Queue Automata*. In: Theory of Computing Systems 65.8 (2021). Full paper of [Köc19], pp. 1211–1242. DOI: 10.1007/s00224-021-10031-2 (cited on p. 174).

[Köc22] C. Köcher. *Rational, Recognizable, and Aperiodic Partially Lossy Queue Languages*. In: International Journal of Algebra and Computation 32.3 (2022). Full paper of [Köc18], pp. 483–528. DOI: 10.1142/S0218196722500230 (cited on p. 78).

Bibliography

[Abd+00] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. *Algorithmic Analysis of Programs with Well Quasi-Ordered Domains*. In: Information and Computation 160.1 (2000), pp. 109–127. DOI: 10.1006/inco.1999.2843 (cited on p. 2).

[Abd+04] P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. *Using Forward Reachability Analysis for Verification of Lossy Channel Systems*. In: Formal Methods in System Design 25.1 (2004), pp. 39–65. DOI: 10.1023/B:FORM.0000033962.51898.1a (cited on pp. 3, 5, 173, 178, 192, 196, 222).

[AJ96a] P. A. Abdulla and B. Jonsson. *Verifying Programs with Unreliable Channels*. In: Information and Computation 127.2 (1996), pp. 91–101. DOI: 10.1006/inco.1996.0053 (cited on pp. 2, 24, 173).

[AJ96b] P. A. Abdulla and B. Jonsson. *Undecidable Verification Problems for Programs with Unreliable Channels*. In: Inf. and Comp. 130.1 (1996), pp. 71–90. DOI: 10.1006/inco.1996.0083 (cited on pp. 2, 173).

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. *Reachability Analysis of Pushdown Automata: Appl. to Model-Checking*. In: CONCUR 1997. Vol. 1243. LNCS. Springer, 1997, pp. 135–150. DOI: 10.1007/3-540-63141-0_10 (cited on pp. 1, 5, 77, 129 sq., 132 sqq., 145 sq., 149, 165, 171, 223 sq.).

[Ber79] J. Berstel. *Transductions and Context-Free Languages*. Teubner Studienbücher, 1979. DOI: 10.1007/978-3-663-09367-1 (cited on pp. 73, 85 sq.).

[BFS20] B. Bollig, A. Finkel, and A. Suresh. *Bounded Reachability Problems Are Decidable in FIFO Machines*. In: CONCUR 2020. Vol. 171. LIPIcs. Dagstuhl Publishing, 2020, 49:1–49:17. DOI: 10.4230/LIPIcs.CONCUR.2020.49 (cited on pp. 3, 174).

[BG99] B. Boigelot and P. Godefroid. *Symbolic Verification of Communication Protocols with Infinite State Spaces Using QDDs*. In: Formal Methods in System Design 14.3 (1999), pp. 237–255. DOI: 10.1023/A:1008719024240 (cited on pp. 173, 175, 178).

[BJW82] R. V. Book, M. Jantzen, and C. Wrathall. *Monadic Thue Systems*. In: Theoretical Computer Science 19.3 (1982), pp. 231–251. DOI: 10.1016/0304-3975(82)90036-6 (cited on p. 76).

[BKM17] B. Bollig, D. Kuske, and R. Mennicke. *The Complexity of Model Checking Multi-Stack Systems*. In: Theory of Computing Systems 60.4 (2017), pp. 695–736. DOI: 10.1007/s00224-016-9700-6 (cited on p. 29).

[BMS82] A. Bertoni, G. Mauri, and N. Sabadini. *Equivalence and Membership Problems for Regular Trace Languages*. In: ICALP 1982. Vol. 140. LNCS. Springer, 1982, pp. 61–71. DOI: 10.1007/BFb0012757 (cited on pp. 152 sq.).

[Boi+97] B. BOIGELOT, P. GODEFROID, B. WILLEMS, and P. WOLPER. *The Power of QDDs*. In: SAS 1997. Vol. 1302. LNCS. Springer, 1997, pp. 172–186. DOI: 10.1007/BFb0032741 (cited on pp. 3, 5, 30, 173 sq., 178, 180, 191 sq., 196, 220, 222, 224).

[Bol06] B. BOLLIG. *Formal Models of Communicating Systems: Languages, Automata, and Monadic Second-Order Logic*. Springer, 2006. DOI: 10.1007/3-540-32923-4 (cited on p. 30).

[Bou+12] P. BOUYER, N. MARKEY, J. OUAKNINE, P. SCHNOEBELEN, and J. WORRELL. *On Termination and Invariance for Faulty Channel Machines*. In: Formal Aspects of Computing 24.4 (2012), pp. 595–607. DOI: 10.1007/s00165-012-0234-7 (cited on p. 26).

[Boz07] L. BOZZELLI. *Complexity Results on Branching-Time Pushdown Model Checking*. In: Theoretical Computer Science 379.1 (2007), pp. 286–297. DOI: 10.1016/j.tcs.2007.03.049 (cited on pp. 2, 129, 134).

[Büc60] J. R. BÜCHI. *Weak Second-Order Arithmetic and Finite Automata*. In: Mathematical Logic Quarterly 6.1-6 (1960), pp. 66–92. DOI: 10.1002/malq.19600060105 (cited on pp. 4, 89, 93, 108, 110, 114 sq., 223).

[BZ83] D. BRAND and P. ZAFIROPULO. *On Communicating Finite-State Machines*. In: Journal of the ACM 30.2 (1983). DOI: 10.1145/322374.322380 (cited on pp. 2, 19, 25, 173).

[CE82] E. M. CLARKE and E. A. EMERSON. *Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic*. In: *Logics of Programs*. Vol. 131. LNCS. Springer, 1982, pp. 52–71. DOI: 10.1007/BFb0025774 (cited on pp. 1, 128).

[CFP96] G. CÉCÉ, A. FINKEL, and S. PURUSHOTAMAN IYER. *Unreliable Channels Are Easier to Verify than Perfect Channels*. In: Information and Computation 124.1 (1996), pp. 20–31. DOI: 10.1006/inco.1996.0003 (cited on p. 26).

[CGK12] A. CYRIAC, P. GASTIN, and K. N. KUMAR. *MSO Decidability of Multi-Pushdown Systems via Split-Width*. In: CONCUR 2012. Vol. 7454. LNCS. Springer, 2012, pp. 547–561. DOI: 10.1007/978-3-642-32940-1_38 (cited on p. 29).

[CL84] M. CLERBOUT and M. LATTEUX. *Partial Commutations and Faithful Rational Transductions*. In: Theoretical Computer Science 34.3 (1984), pp. 241–254. DOI: 10.1016/0304-3975(84)90053-7 (cited on p. 27).

[Cle86] J. C. CLEAVELAND. *An Introduction to Data Types*. Addison-Wesley, 1986 (cited on p. 13).

[CO22] W. CZERWIŃSKI and Ł. ORLIKOWSKI. *Reachability in Vector Addition Systems Is Ackermann-complete*. In: FOCS 2021. IEEE, 2022, pp. 1229–1240. DOI: 10.1109/FOCS52979.2021.00120 (cited on pp. 2, 174, 219).

[Cor+09] T. H. CORMEN, C. E. LEISERSON, R. E. RIVEST, and C. STEIN. *Introduction to Algorithms*. 3rd ed. MIT Press, 2009. 228-355 (cited on p. 13).

[CP61] A. H. CLIFFORD and G. B. PRESTON. *The Algebraic Theory of Semigroups*. Vol. 1. Mathematical Surveys and Monographs 7. AMS, 1961. DOI: 10.1090/surv/007.1 (cited on pp. 90 sq.).

[CP85] R. Cori and D. Perrin. *Automates et Commutations Partielles*. In: RAIRO - Informatique théorique 19.1 (1985), pp. 21–32. DOI: 10.1051/ita/1985190100211 (cited on p. 27).

[CR86] M. Chrobak and W. Rytter. *Unique Decipherability for Partially Commutative Alphabet*. In: MFCS 1986. Vol. 233. LNCS. Springer, 1986, pp. 256–263. DOI: 10.1007/BFb0016249 (cited on pp. 73, 81, 83).

[CS08] P. Chambart and P. Schnoebelen. *The Ordinal Recursive Complexity of Lossy Channel Systems*. In: LICS 2008. IEEE Computer Society Press, 2008, pp. 205–216. DOI: 10.1109/LICS.2008.47 (cited on pp. 2, 22, 24, 173, 192).

[Die90] V. Diekert. *Combinatorics on Traces*. Vol. 454. LNCS. Springer, 1990. DOI: 10.1007/3-540-53031-2 (cited on pp. 28, 44, 225).

[DM97] V. Diekert and Y. Métivier. "Partial Commutation and Traces". In: *Handbook of Formal Languages: Volume 3 Beyond Words*. Ed. by G. Rozenberg and A. Salomaa. Springer, 1997, pp. 457–533. DOI: 10.1007/978-3-642-59126-6_8 (cited on p. 28).

[DR95] V. Diekert and G. Rozenberg. *The Book of Traces*. World scientific, 1995. DOI: 10.1142/9789814261456 (cited on pp. 2, 27, 138, 225).

[Ebi94] W. Ebinger. *Charakterisierung von Sprachklassen unendlicher Spuren durch Logiken*. Dissertation. Universität Stuttgart, 1994 (cited on p. 89).

[Ebi95] W. Ebinger. "Logical Definability of Trace Languages". In: *The Book of Traces*. Ed. by V. Diekert and G. Rozenberg. World scientific, 1995. DOI: 10.1142/9789814261456_0010 (cited on pp. 89, 108).

[EH83] E. A. Emerson and J. Y. Halpern. *"Sometimes" and "Not Never" Revisited: On Branching versus Linear Time*. In: POPL 1983. New York, NY, USA: Association for Computing Machinery, 1983, pp. 127–140. DOI: 10.1145/567067.567081 (cited on p. 128).

[Ehr61] A. Ehrenfeucht. *An Application of Games to the Completeness Problem for Formalized Theories*. In: Fundamenta Mathematicae 49.129-141 (1961), p. 13. DOI: 10.4064/fm-49-2-129-141 (cited on p. 11).

[Esp+00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. *Efficient Algorithms for Model Checking Pushdown Systems*. In: CAV 2000. Vol. 1855. LNCS. Springer, 2000, pp. 232–247. DOI: 10.1007/10722167_20 (cited on pp. 130, 132).

[EZ74] A. Ehrenfeucht and P. Zeiger. *Complexity Measures for Regular Expressions*. In: *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*. STOC 1974. New York, NY, USA: Association for Computing Machinery, 1974, pp. 75–79. DOI: 10.1145/800119.803886 (cited on p. 115).

[FP20] A. Finkel and M. Praveen. *Verification of Flat FIFO Systems*. In: Logical Methods in Computer Science 16.4 (2020). DOI: 10.23638/LMCS-16(4:4)2020 (cited on pp. 3, 174, 222).

[Fra54] R. Fraïssé. *Sur Quelques Classifications Des Systèmes de Relations*. In: Université d'Alger, Publications Scientifiques, Série A 1 (1954), pp. 35–182 (cited on p. 11).

[FS01] A. FINKEL and P. SCHNOEBELEN. *Well-Structured Transition Systems Everywhere!* In: Theoretical Computer Science. ISS 256.1 (2001), pp. 63–92. DOI: 10.1016/S0304-3975(00)00102-X (cited on p. 2).

[FS02] H. FERNAU and R. STIEBE. *Sequential Grammars and Automata with Valences*. In: Theoretical Computer Science 276.1 (2002), pp. 377–405. DOI: 10.1016/S0304-3975(01)00282-1 (cited on p. 14).

[FWW97] A. FINKEL, B. WILLEMS, and P. WOLPER. *A Direct Symbolic Approach to Model Checking Pushdown Systems*. In: Electronic Notes in Theoretical Computer Science. Infinity 1997 9 (1997), pp. 27–37. DOI: 10.1016/S1571-0661(05)80426-8 (cited on pp. 1, 5, 77, 130, 132 sq., 151, 158, 162, 171, 223 sq.).

[Gil96] R. H. GILMAN. *Formal Languages and Infinite Groups*. In: Discrete Mathematics and Theoretical Computer Science 25 (1996), pp. 27–51 (cited on p. 38).

[GK07] P. GASTIN and D. KUSKE. *Uniform Satisfiability in PSPACE for Local Temporal Logics Over Mazurkiewicz Traces*. In: Fundamenta Informaticae 80.1-3 (2007), pp. 169–197 (cited on pp. 126, 170).

[GR86] A. GIBBONS and W. RYTTER. *On the Decidability of Some Problems about Rational Subsets of Free Partially Commutative Monoids*. In: Theoretical Computer Science 48 (1986), pp. 329–337. DOI: 10.1016/0304-3975(86)90101-5 (cited on pp. 73, 85 sq.).

[Gre78] S. A. GREIBACH. *Remarks on Blind and Partially Blind One-Way Multicounter Machines*. In: Theoretical Computer Science 7 (1978), pp. 311–324 (cited on p. 16).

[GRS91] G. GUAIANA, A. RESTIVO, and S. SALEMI. *On Aperiodic Trace Languages*. In: STACS 1991. Vol. 480. LNCS. Springer, 1991, pp. 76–88. DOI: 10.1007/BFb0020789 (cited on p. 117).

[Hai69] L. H. HAINES. *On Free Monoids Partially Ordered by Embedding*. In: Journal of Combinatorial Theory 6.1 (1969), pp. 94–98. DOI: 10.1016/S0021-9800(69)80111-0 (cited on pp. 135, 173).

[Heu+10] A. HEUSSNER, J. LEROUX, A. MUSCHOLL, and G. SUTRE. *Reachability Analysis of Communicating Pushdown Systems*. In: vol. 6014. LNCS. Springer, 2010, pp. 267–281. DOI: 10.1007/978-3-642-12032-9_19 (cited on p. 29).

[HKZ17] M. HUSCHENBETT, D. KUSKE, and G. ZETZSCHE. *The Monoid of Queue Actions*. In: Semigroup forum 95.3 (2017), pp. 475–508. DOI: 10.1007/s00233-016-9835-4 (cited on pp. 19, 35, 47 sqq., 65, 70, 78, 81, 94, 96 sq., 111).

[HMU01] J. E. HOPCROFT, R. MOTWANI, and J. D. ULLMAN. *Introduction to Automata Theory, Languages, and Computation*. 2nd ed. Addison Wesley, 2001. 560 pp. (cited on pp. 18, 29).

[How95] J. M. HOWIE. *Fundamentals of Semigroup Theory*. London Mathematical Society Monographs. Oxford, New York: Oxford University Press, 1995. 362 pp. (cited on p. 41).

[HSS14] C. HAASE, S. SCHMITZ, and P. SCHNOEBELEN. *The Power of Priority Channel Systems*. In: Logical Methods in Computer Science 10 (4:4 2014). DOI: 10.2168/LMCS-10(4:4)2014 (cited on pp. 2, 20).

[Hut+18] M. Hutagalung, N. Hundeshagen, D. Kuske, M. Lange, and É. Lozes. *Multi-Buffer Simulations: Decidability and Complexity*. In: Information and Computation. GandALF 2016 262 (2018), pp. 280–310. DOI: 10.1016/j.ic.2018.09.008 (cited on p. 30).

[Jon75] N. D. Jones. *Space-Bounded Reducibility among Combinatorial Problems*. In: Journal of Computer and System Sciences 11.1 (1975), pp. 68–85. DOI: 10.1016/S0022-0000(75)80050-X (cited on p. 80).

[Kam09] M. Kambites. *Formal Languages and Groups as Memory*. In: Communications in Algebra 37.1 (2009), pp. 193–208. DOI: 10.1080/00927870802243580 (cited on p. 35).

[Kam68] J. Kamp. *Tense Logic and the Theory of Linear Order*. Dissertation. Los Angeles: University of California at Los Angeles, 1968 (cited on pp. 117, 119).

[Kle51] S. C. Kleene. *Representation of Events in Nerve Nets and Finite Automata*. DTIC Document, 1951 (cited on pp. 4, 73, 75, 93, 103, 115, 223).

[KM21] D. Kuske and A. Muscholl. "Communicating Automata". In: *Handbook of Automata*. Vol. 2. EMS Print, 2021, pp. 1147–1188. DOI: 10.4171/Automata (cited on pp. 2, 30, 173).

[KM69] R. M. Karp and R. E. Miller. *Parallel Program Schemata*. In: Journal of Computer and System Sciences 3.2 (May 1, 1969), pp. 147–195. DOI: 10.1016/S0022-0000(69)80011-5 (cited on p. 16).

[Köc16] C. Köcher. *Einbettungen in das Transformationsmonoid einer vergesslichen Warteschlange*. Master's Thesis. Ilmenau: Technische Universität Ilmenau, 2016 (cited on pp. 4, 25, 35, 50 sq., 60).

[Ler22] J. Leroux. *The Reachability Problem for Petri Nets Is Not Primitive Recursive*. In: FOCS 2021. IEEE, 2022, pp. 1241–1252. DOI: 10.1109/FOCS52979.2021.00121 (cited on pp. 2, 174, 219).

[Lev44] F. W. Levi. *On Semigroups*. In: Bulletin of the Calcutta Mathematical Society 36 (1944), pp. 141–146 (cited on p. 28).

[Lib04] L. Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer, 2004. DOI: 10.1007/978-3-662-07003-1 (cited on p. 115).

[LN11] S. La Torre and M. Napoli. *Reachability of Multistack Pushdown Systems with Scope-Bounded Matching Relations*. In: CONCUR 2011. Vol. 6901. LNCS. Springer, 2011, pp. 203–218. DOI: 10.1007/978-3-642-23217-6_14 (cited on p. 29).

[Loh13] M. Lohrey. *The Rational Subset Membership Problem for Groups: A Survey*. In: *Groups St Andrews*. Vol. 422. London Mathematical Society Lecture Note Series. Cambridge University Press, 2013, pp. 368–389. DOI: 10.1017/CBO9781316227343.024 (cited on p. 73).

[Lot02] M. Lothaire. *Algebraic Combinatorics on Words*. Encyclopedia of Mathematics and Its Applications. Cambridge University Press, 2002. DOI: 10.1017/CBO9781107326019 (cited on p. 16).

[LS19] J. Leroux and S. Schmitz. *Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension*. In: LICS 2019. 2019, pp. 1–13. DOI: 10.1109/LICS.2019.8785796 (cited on p. 219).

[Lya53] E. S. LYAPIN. *The canonical form of elements of an associative system given by defining relations*. In: Leningrad. Gos. Ped. Inst. Učen. Zap 89 (1953), pp. 45–54 (cited on p. 16).

[May03] R. MAYR. *Undecidable Problems in Unreliable Computations*. In: Theoretical Computer Science. Latin American Theoretical Informatics 297.1 (2003), pp. 337–354. DOI: 10.1016/S0304-3975(02)00646-1 (cited on p. 173).

[May84] E. MAYR. *An Algorithm for the General Petri Net Reachability Problem*. In: SIAM Journal on Computing 13.3 (1984), pp. 441–460. DOI: 10.1137/0213029 (cited on pp. 2, 219).

[Maz77] A. MAZURKIEWICZ. *Concurrent Program Schemes and Their Interpretations*. In: DAIMI Report Series 6.78 (1977) (cited on pp. 2, 27).

[McK64] J. D. MCKNIGHT. *Kleene Quotient Theorems*. In: Pacific Journal of Mathematics 14.4 (1964), pp. 1343–1352 (cited on pp. 73, 75, 83 sq., 89).

[Min67] M. L. MINSKY. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967 (cited on pp. 2, 17).

[MP71] R. MCNAUGHTON and S. A. PAPERT. *Counter-Free Automata*. Vol. 65. The MIT Press, 1971 (cited on pp. 5, 75, 93, 99, 102, 117, 119).

[Mus94] A. MUSCHOLL. *Über die Erkennbarkeit unendlicher Spuren*. Vol. 17. Teubner-Texte Zur Informatik. Stuttgart: Teubner Verlagsgesellschaft, 1994. 114 pp. DOI: 10.1007/978-3-322-95371-1 (cited on p. 168).

[NP70] M. NIVAT and J.-F. PERROT. *Une Généralisation Du Monoïde Bicyclique*. In: Comptes Rendus de l'Académie des Sciences de Paris 271 (1970), pp. 824–827 (cited on pp. 18, 36).

[Och85] E. OCHMAŃSKI. *Regular Behaviour of Concurrent Systems*. In: Bulletin of the EATCS 27 (1985), pp. 56–67 (cited on pp. 89, 103, 225).

[Pău80] G. PĂUN. *A New Generative Device: Valence Grammars*. In: Revue Roumaine de Mathématiques Pures et Appliquées 25 (1980), pp. 911–924 (cited on p. 14).

[Pet62] C. A. PETRI. *Kommunikation Mit Automaten*. Dissertation. Bonn: Universität Bonn, 1962 (cited on p. 16).

[Pin10] J.-É. PIN. *Mathematical Foundations of Automata Theory*. In: Lecture notes LIAFA, Université Paris 7 (2010) (cited on pp. 73 sqq.).

[Pnu77] A. PNUELI. *The Temporal Logic of Programs*. In: FOCS 1977. 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32 (cited on pp. 1, 128).

[Pos46] E. L. POST. *A Variant of a Recursively Unsolvable Problem*. In: Bulletin of the American Mathematical Society 52.4 (1946), pp. 264–268 (cited on pp. 83, 223).

[RK09] E. RENDER and M. KAMBITES. *Rational Subsets of Polycyclic Monoids and Valence Automata*. In: Information and Computation 207.11 (2009), pp. 1329–1339. DOI: 10.1016/j.ic.2009.02.012 (cited on pp. 4, 35, 73, 75 sq., 87, 223).

[Sch02] P. SCHNOEBELEN. *Verifying Lossy Channel Systems Has Nonprimitive Recursive Complexity*. In: Information Processing Letters 83.5 (2002), pp. 251–261. DOI: 10.1016/S0020-0190(01)00337-4 (cited on pp. 2, 24, 173, 192).

[Sch19] T. Schellmann. *Model-Checking von Kellersystemen*. Bachelor's Thesis. Ilmenau: Technische Universität Ilmenau, 2019 (cited on p. 132).

[Sch20] P. Schnoebelen. *On Flat Lossy Channel Machines*. 2020. arXiv: 2007.05269 [cs] (cited on pp. 3, 174, 222).

[Sch65] M. P. Schützenberger. *On Finite Monoids Having Only Trivial Subgroups*. In: Information and control 8.2 (1965), pp. 190–194. DOI: 10.1016/S0019-9958(65)90108-7 (cited on pp. 5, 75, 117, 119).

[SM73] L. J. Stockmeyer and A. R. Meyer. *Word Problems Requiring Exponential Time*. In: *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*. STOC 1973. New York, NY, USA: Association for Computing Machinery, 1973, pp. 1–9. DOI: 10.1145/800125.804029 (cited on p. 115).

[SP53] A. A. Sardinas and G. W. Patterson. *A Necessary and Sufficient Condition for Unique Decomposition of Coded Messages*. In: *Proceedings of the Institute of Radio Engineers*. Vol. 41. 3. 1953, pp. 425–425 (cited on p. 77).

[SS20] G. Saake and K.-U. Sattler. *Algorithmen und Datenstrukturen: Eine Einführung mit Java*. 6th ed. dpunkt.verlag, 2020 (cited on pp. 13, 17).

[Thi94] P. S. Thiagarajan. *A Trace Based Extension of Linear Time Temporal Logic*. In: LICS 1994. 1994, pp. 438–447. DOI: 10.1109/LICS.1994.316047 (cited on p. 168).

[Tho82] W. Thomas. *Classifying Regular Events in Symbolic Logic*. In: Journal of Computer and System Sciences 25.3 (1982), pp. 360–376. DOI: 10.1016/0022-0000(82)90016-2 (cited on p. 120).

[VW86] M. Y. Vardi and P. Wolper. *An Automata-Theoretic Approach to Automatic Program Verification*. In: LICS 1986. Cambridge, 1986, pp. 322–331 (cited on pp. 1, 5, 127 sq.).

[Wal00] I. Walukiewicz. *Model Checking CTL Properties of Pushdown Systems*. In: FSTTCS 2000. Vol. 1974. LNCS. Springer, 2000, pp. 127–138. DOI: 10.1007/3-540-44450-5_10 (cited on pp. 1, 129, 134).

[Wal01] I. Walukiewicz. *Pushdown Processes: Games and Model-Checking*. In: Information and Computation 164.2 (2001), pp. 234–263. DOI: 10.1006/inco.2000.2894 (cited on pp. 1, 129, 134).

[Wal98] I. Walukiewicz. *Difficult Configurations – On the Complexity of LTrL*. In: ICALP 1998. Vol. 1443. LNCS. Springer, 1998, pp. 140–151. DOI: 10.1007/BFb0055048 (cited on p. 170).

[Zet16] G. Zetzsche. *Monoids as Storage Mechanisms*. Dissertation. Kaiserslautern: Technische Universität Kaiserslautern, 2016 (cited on p. 35).

[Zie87] W. Zielonka. *Notes on Finite Asynchronous Automata*. In: RAIRO - Theoretical Informatics and Applications 21.2 (1987), pp. 99–135. DOI: 10.1051/ita/198721020099 1 (cited on p. 138).

Glossary

Abbreviations

cf.	confer (*compare*)
DFA	deterministic finite automaton
e.g.	exempli gratia (*for example*)
et al.	et alii / et aliae (*and others*)
fifo	first in, first out
i.e.	id est (*that is*)
i.h.	induction hypothesis
iff	if, and only if / $\iff$
lifo	last in, first out
NFA	nondeterministic finite automaton
PDA	pushdown automaton
plq	partially lossy queue
pls	partially lossy stack
resp.	respectively
w.l.o.g.	without loss of generality
wrt.	with respect to

Symbols and Notations

$C(\mathfrak{A})$	accepted configurations of $\mathfrak{A}$ 119, 133
$L(\mathfrak{A})$	accepted language of $\mathfrak{A}$ 9
$L^\omega(\mathfrak{A})$	accepted ω-language of $\mathfrak{A}$ 117
$T(\mathfrak{A})$	accepted trace language of $\mathfrak{A}$ 126
Ω_ℓ	action sequences with large overlap's bounded width 85
$\vec{\boldsymbol{p}} \xrightarrow{\lambda}_{\mathfrak{A}} \vec{\boldsymbol{q}}$	asynchronous run labeled with λ 127
$\mathfrak{A}_{P_1 \to P_2}$	automaton $\mathfrak{A}$ accepting runs from P_1 to P_2 9
$\textsc{BackReach}_{\mathcal{D}}(L, T)$	backwards reachable contents 115

$\equiv_{\mathcal{D}}, \equiv$	behavioral equivalence in $\mathcal{D}$ . . . 33
$\chi(t)$	characteristic . . . 45, 51
$[S]_{\equiv}$	closure under equivalence class $\equiv$ . . . 7
CTL^*	combination of CTL and LTL . . . 116
$\mathcal{G}^c$	complementary graph of $\mathcal{G}$ . . . 8
CTL	computation tree logic . . . 116
$\mathcal{G}_{\mathfrak{A}}$	configuration graph of $\mathfrak{A}$ . . . 9, 14
$\mathcal{G}_{\mathcal{A}}$	dependence graph of $\mathcal{A}$ . . . 26
$\mathcal{Q}_{\mathcal{A}}$	distributed queue . . . 29
$\mathcal{P}_{\mathcal{A}}$	distributed stack / pushdown . . . 28
$\downarrow_{\leq} S$	downwards closure wrt. quasi ordering $\leq$ . . . 7
$\mathrm{d}(t)$	duality map . . . 43
$A \hookrightarrow B$	embedding (injective homomorphism) from A into B . . . 56
$A \twoheadrightarrow B$	epimorphism (surjective homomorphism) from A into B . . . 26
$\bot$	error state . . . 13
$\mathsf{FO}[\tau]$	first-order logic on τ . . . 10
$\mathrm{post}^*_{\mathcal{M}}(C), \mathrm{pre}^*_{\mathcal{M}}(C)$	forwards and backwards reachable nodes / configurations . . . 114
$\textsc{Reach}_{\mathcal{D}}(L, T)$	forwards reachable contents . . . 115
$\mathrm{fv}(\phi)$	free variables of a formula ϕ . . . 10
L, R, J, H, D	Green's relations . . . 8
$a \parallel b$	independent letters a and b . . . 26
$w[i, j]$	indexed infix . . . 8
$\mathrm{Alph}(\lambda)$	induced alphabet of a trace λ . . . 27
$\mathcal{M}(\mathfrak{S})$	induced Kripke-structure from a $\mathcal{D}$-system $\mathfrak{S}$ . . . 113
$\sqsubseteq_{\mathcal{L}}$	$\mathcal{L}$-subword . . . 21
$T(\phi)$	language of all transformations satisfying ϕ . . . 100
$L(\phi)$	language of all words satisfying ϕ . . . 84
$s\,R, R\,t$	left and right restrictions of binary relations . . . 7
LTL	linear (or propositional) temporal logic . . . 116
TrPTL	linear temporal logic on traces . . . 154
$\mathsf{MSO}[\tau]$	monadic second-order logic on τ . . . 10
$\zeta_{\mathcal{A}}(w)$	natural epimorphism of the trace monoid $\mathbb{M}(\mathcal{A})$ . . . 26
$\eta_{\mathcal{D}}(t), \eta(t)$	natural epimorphism of the transformation monoid $\mathbb{T}(\mathcal{D})$ . . . 34
$\mathrm{nf}(t)$	normal form . . . 37, 50
$\mathrm{rd}_2(t)$	overlap of a transformation sequence . . . 51
$\mathrm{obw}(t)$	overlap's bounded width . . . 85
C_k	(partially blind) counters . . . 16
$\mathcal{PLQ}_{\mathcal{L}}$	partially lossy queue with default semantics . . . 23
$\mathcal{Q}_{\mathcal{L}}$	partially lossy queue with read-lossy semantics . . . 23

Miscellaneous Definitions