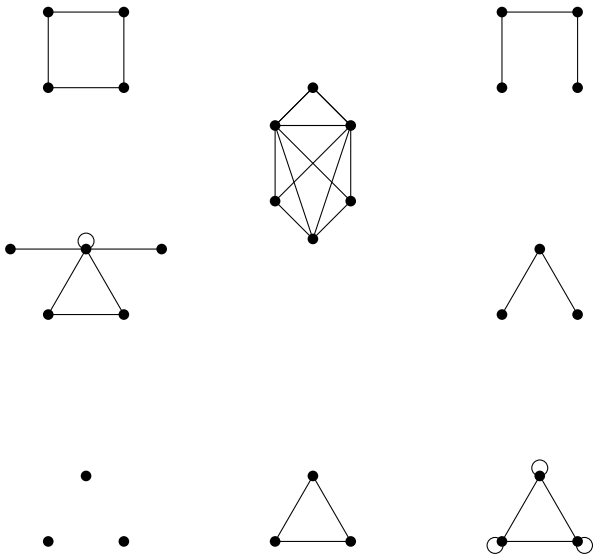
Monoids as Storage Mechanisms

Georg Zetzsche



Vom Fachbereich Informatik der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades Doktor der Naturwissenschaften
(Dr. rer. nat.) genehmigte Dissertation

Dekan:	Prof. Dr. Klaus Schneider
Vorsitzender der Prüfungskommission:	Prof. Dr. Jens Schmitt
1. Berichterstatter	Prof. Dr. Roland Meyer
2. Berichterstatter	Prof. Dr. Markus Lohrey
3. Berichterstatter	Philippe Schnoebelen

Datum der wissenschaftlichen Aussprache: 19. Juni 2015

D 386

Abstract

Automata theory has given rise to a variety of automata models that consist of a finite-state control and an infinite-state storage mechanism. The aim of this work is to provide insights into how the structure of the storage mechanism influences the expressiveness and the analyzability of the resulting model. To this end, it presents generalizations of results about individual storage mechanisms to larger classes. These generalizations characterize those storage mechanisms for which the given result remains true and for which it fails.

In order to speak of classes of storage mechanisms, we need an overarching framework that accommodates each of the concrete storage mechanisms we wish to address. Such a framework is provided by the model of *valence automata*, in which the storage mechanism is represented by a monoid. Since the monoid serves as a parameter to specifying the storage mechanism, our aim translates into the question: *For which monoids does the given (automata-theoretic) result hold?*

As a first result, we present an algebraic characterization of those monoids over which valence automata accept *only regular languages*. In addition, it turns out that for each monoid, this is the case if and only if *valence grammars*, an analogous grammar model, can generate only context-free languages.

Furthermore, we are concerned with closure properties: We study which monoids result in a *Boolean closed* language class. For every language class that is closed under rational transductions (in particular, those induced by valence automata), we show: If the class is Boolean closed and contains any non-regular language, then it already includes the whole arithmetical hierarchy.

This work also introduces the class of *graph monoids*, which are defined by finite graphs. By choosing appropriate graphs, one can realize a number of prominent storage mechanisms, but also combinations and variants thereof. Examples are pushdowns, counters, and Turing tapes. We can therefore relate the structure of the graphs to computational properties of the resulting storage mechanisms.

In the case of graph monoids, we study (i) the decidability of the *emptiness problem*, (ii) which storage mechanisms guarantee *semilinear Parikh images*, (iii) when *silent transitions* (i.e. those that read no input) can be avoided, and (iv) which storage mechanisms permit the computation of *downward closures*.

Acknowledgements

First and foremost, I would like to thank my adviser Roland Meyer. I am deeply grateful for his interest in my work, the new perspectives I gained from discussions with him, and for all I have learned from him. I feel extremely lucky to have had such a supportive adviser who was always available when I sought advice.

I am also greatly indebted to Markus Lohrey. In Zürich in 2011, he introduced me to graph groups, which was certainly a decisive inspiration for this work. Our ensuing collaboration (on topics of this thesis and related ones) and everything I learned from him benefited my work substantially.

Moreover, I feel honored and grateful that Markus Lohrey and Philippe Schnoebelen agreed to take the time to review this thesis.

Furthermore, I am thankful to my other respected coauthors during my time as a doctoral student. In 2012, I had the opportunity to spend one week in New York to work with Benjamin Steinberg and I am grateful for all the stimulating discussions we had and the challenging questions he raised. Then in 2013, upon the kind invitation of Martin Huschenbett and Dietrich Kuske, I could spend two weeks doing exciting research in Ilmenau. I am thankful for the time they spent with me and the many things I learned from them.

I was fortunate enough to have two great students write their Bachelor thesis under my supervision, Phoebe Buckheister (who also became a coauthor) and Martin Köhler. I am indebted to them for the many discussions we had and particularly the questions they asked.

Moreover, I am grateful to have been part of the Concurrency Theory Group in Kaiserslautern with Georgel Călin, Egor Derevenetc, Florian Furbach, Reiner Hüchting, Klaus Madlener, and Roland Meyer. It was a pleasure to work in this group, which fosters scientific curiosity and open-mindedness. I fondly remember our lively and entertaining after-lunch debates. I am especially thankful to Reiner Hüchting for all his help and our countless scientific and non-scientific discussions.

Before I came to Kaiserslautern, I studied at Universität Hamburg. Of course, the education I received in the group "Theoretische Grundlagen der Informatik" was essential for this work. I am therefore deeply grateful to Lawrence Cabac, Michael Duvigneau, Frank Heitmann, Michael Köhler-Bußmeier, Daniel Moldt, Berndt Müller, Heiko Rölke and Rüdiger Valk for the theory they have taught me and particularly Matthias Jantzen and Manfred Kudlek (who sadly passed away in June 2012) for acquainting me with formal languages.

I am immensely thankful to my wonderful family. Without their help, I would have had none of the opportunities that allowed me to pursue this work. I would like to thank my parents Albert and Claudia and my siblings Anke, Björn, Jakob, Maria, Martin, and Ruth for all of their love and encouragement. Finally, I am deeply grateful to my amazing Katie, whose endless support I can only hope to be able to repay.

Contents

Chapter 1

Introduction

1.1 Monoids as storage mechanisms

One of the main purposes of theoretical computer science is to understand the principles of computation. In the field of automata theory, this goal is pursued by studying mathematical models of computing devices with regard to what behavior they can exhibit and what we can infer about such a device when given a description.

These two types of questions each have their own motivation. The first type addresses *expressiveness*. This aspect is important to understand because it explains what we can compute with limited resources and what systems we can describe with the respective models. The second type of questions explores the *analyzability* of models. This perspective is instrumental when we want to algorithmically verify properties of systems, which, due to the advent of increasingly complex and concurrent systems, has become a task of significant weight.

The perspectives of expressiveness and analyzability are deeply intertwined: They are conflicting qualities insofar as the more expressive a model is, the more difficult it usually is to analyze. For these reasons, it has become a strong driving force of today's research in theoretical computer science to understand how we can provide models that are expressive enough for a given type of systems and yet are simple enough to be amenable to analysis.

In a tradition initiated by Turing in the introduction of the eponymous machine, automata theory yielded a rich variety of models that comprise a finite-state control and a potentially infinite data repository. The models are obtained by imposing restrictions on how the data can be stored, manipulated, and retrieved, while permitting arbitrary use of the finite-state control. In terms of hard- and software systems that can be represented by such models, this means we can precisely reflect arbitrary control flows, but we abstract from certain aspects of data access. For example, pushdown automata can correctly imitate the control flow and calling stack of a recursive program, but heap memory cannot be represented. A form of data repository, together with the permitted modes of access, is called a *storage mechanism*. Examples of storage mechanisms include Turing machine tapes, pushdown storages, and various kinds of counters.

Instead of investigating the properties of a concrete model of computation, the present work attempts to provide general insights on how expressiveness

and analyzability of a model of computation are affected by the structure of the storage mechanism. To this end, it presents generalizations of results about concrete storage mechanisms to larger classes of storage mechanisms. These generalizations will characterize those storage mechanisms for which the given result remains true and for which it fails.

Storage mechanisms as monoids In order to speak of classes of storage mechanisms, we need an overarching framework that accommodates each of the concrete storage mechanisms we wish to address. Such a framework is provided by interpreting storage mechanisms as *monoids*. Suppose a storage mechanism consists of a (potentially infinite) set of states, a finite set of functions representing its available operations, an initial state, and a collection of valid final states. To account for operations that are not always applicable, such as a pop operation for a stack symbol that is not currently at the top, the functions can be partial functions. For example, a pushdown storage with stack alphabet Γ consists of the set Γ^* as its set of states, the operations push_a and pop_a for each $a \in \Gamma$, and the empty word ε as its initial state and its final state (assuming that it accepts with an empty stack). As partial functions, the operations push_a and pop_a are defined as

$$\begin{aligned} \mathsf{push}_a \colon \Gamma^* &\to \Gamma^*, & \mathsf{pop}_a \colon \Gamma^* &\nrightarrow \Gamma^*, \\ w &\mapsto wa & wa &\mapsto w. \end{aligned}$$

(here, we denote partial functions by $\nrightarrow$). Note that pop_a is defined on precisely those words that end in a. Another example is the Minsky counter, which has $\mathbb{N}$, the set of natural numbers, as its set of states and has inc (*increment*), dec (*decrement*), and zero (*zero test*) as its operations:

$$\begin{aligned} \mathsf{inc} \colon \mathbb{N} &\to \mathbb{N}, & \mathsf{dec} \colon \mathbb{N} &\nrightarrow \mathbb{N}, & \mathsf{zero} \colon \mathbb{N} &\nrightarrow \mathbb{N}, \\ n &\mapsto n+1, & n &\mapsto n-1, & 0 &\mapsto 0. \end{aligned}$$

Note that here, the decrement operation is undefined for state 0 and the zero test operation is defined only in state 0.

To such a storage mechanism, we can associate the monoid of all compositions of available operations. Let us examine what this yields in the case of a pushdown store as above. If we compose push_a and pop_b for $a \neq b$, we obtain the function 0, which is defined nowhere: After pushing an a, popping b cannot be defined. Moreover, composing 0 with any other operation yields 0 again. If, however, we only consider compositions where such incompatible push and pop do not occur, the reader can verify that we always get functions of the form $P_{u,v}$ for $u, v \in \Gamma^*$, where

$$\begin{aligned} P_{u,v} \colon \Gamma^* &\nrightarrow \Gamma^*, \\ wu &\mapsto wv, \end{aligned}$$

is defined on precisely those words with suffix u. Therefore, the resulting monoid has the elements $\{0\} \cup \{P_{u,v} \mid u, v \in \Gamma^*\}$. Let us consider the case of a Minsky counter. Any composition of just the increment and decrement operations yields an element $C_{r,s}$ such that

$$\begin{aligned} C_{r,s} \colon \mathbb{N} &\nrightarrow \mathbb{N}, \\ n + r &\mapsto n + s, \end{aligned}$$

which is defined on all numbers $\geqslant r$. If the composition involves a zero test, then it is either 0 as above or it is defined on only one element $r \in \mathbb{N}$ and of the form $D_{r,s}$, for which

$$\begin{aligned} D_{r,s} \colon \mathbb{N} &\nrightarrow \mathbb{N}, \\ r &\mapsto s. \end{aligned}$$

Hence, the corresponding monoid comprises the set $\{0, C_{r,s}, D_{r,s} \mid r, s \in \mathbb{N}\}$.

Monoids as storage mechanisms The advantage of interpreting storage mechanisms as monoids is that we can go in the other direction and interpret *monoids as storage mechanisms*: The elements of the monoid determine the set of states as well as the set of operations and the identity element is the final state. This allows us to use algebraic constructions to synthesize similar storage mechanisms and thus identify *what structural traits of the mechanism are responsible for which computational properties*. For example, we will define monoids by graphs that may contain loops. We will then see that graphs with no loops or edges correspond to pushdown storages. If the graph has no loops, but is otherwise a clique, it is equivalent to counters without zero tests (that cannot go below zero). This is usually called a set of *partially blind counters*. Moreover, if the graph is a clique and has loops everywhere, we obtain counters that can go below zero and are only zero tested in the end of the computation, hence a set of *blind counters*. See Section 2.4 for details and more examples.

This means we can regard these concrete storage as points on a spectrum and examine where exactly the computational properties remain true and where they cease to hold. For example, it is known that automata with a pushdown or with blind counters accept languages with semilinear Parikh images, which is not true of partially blind counters. We can now study which graphs exactly guarantee semilinearity of the accepted languages (see Chapter 7 for a characterization).

Valence automata We investigate monoids as storage mechanisms by deploying them in the framework of *valence automata*. A valence automaton over a monoid M is a finite automaton in which each edge carries an input word and an element of M. The language accepted by such an automaton consists of those words spelled by paths whose composition of monoid elements is the identity.

Valence automata are not a new concept and have been studied before by several authors from various perspectives (see Section 2.9 for an overview). What distinguishes this work from earlier ones is that it systematically generalizes results for concrete models of automata with storage. Specifically, for each of a series of results about concrete storage mechanisms, it presents a broader class of monoids and identifies those members of the class to which the result carries over.

1.2 Outline and main contributions

Let us give an overview of the main contributions of this work. Since the results of this thesis roughly correspond to the chapters, this also serves as an outline of the structure of this work.

Graph monoids In order to generalize statements about concrete storage mechanisms, we present a class of monoids that accommodates a number of well-known storage mechanisms. These monoids are defined by graphs and are therefore dubbed *graph monoids*. In addition to the abovementioned pushdown automata, partially blind multicounter automata, and blind multicounter automata, by choosing appropriate graphs, one can also realize Turing machine tapes and combinations of all these mechanisms, such as pushdowns *with partially blind counters*. As part of Chapter 2, which introduces the main concepts needed in later chapters, Section 2.4 introduces graph monoids.

Increasing expressiveness Chapter 3 presents an algebraic characterization of those monoids that *increase the expressiveness* in the following sense: Without the storage mechanism, finite automata only accept regular languages. Hence, we describe those monoids M for which valence automata over M can accept non-regular languages. In fact, we show that this also characterizes those monoids for which deterministic valence automata are expressively weaker and those for which valence grammars can generate non-context-free languages. Valence grammars are a concept related to valence automata and equip context-free grammars with a monoid storage. While the characterization of monoids that increase expressiveness in valence automata has been obtained independently by Render in her thesis [**Render2010**], the latter characterization for valence grammars answers an open problem posed by Fernau and Stiebe [**FernauStiebe2002a**].

Emptiness problem In Chapter 4, we turn to the decidability of the *emptiness problem*. Using graph monoids, one can realize a pushdown storage with partially blind counters, for which the decidability of the emptiness problem remains a long-standing open question [**Reinhardt2008**].

However, if we forbid the subgraphs corresponding to these mechanisms, we can characterize those with a decidable emptiness problem. The result generalizes the decidability for pushdown automata and for partially blind counter automata (or equivalently, Petri nets). Moreover, this extends a result of **LohreySteinberg2008**, which characterizes those graph groups with a decidable rational subset membership problem. Where **LohreySteinberg2008** rely on semilinearity arguments, we use a reduction to the reachability problem of priority multicounter machines, which has been proven decidable by **Reinhardt2008** [**Reinhardt2008**].

Boolean closure Chapter 5 is concerned with closure properties of the languages accepted by valence automata. Since it is well-known that the regular languages are closed under the Boolean operations (union, intersection, and complementation), we ask for which monoids M, the class of languages accepted by valence automata over M is *closed under the Boolean operations*. Our result is a very negative answer and goes beyond valence automata. It is shown here that every language class that is closed under the Boolean operations and rational transductions and contains an arbitrary non-regular language already includes the whole arithmetical hierarchy. It follows in particular that every language class induced by valence automata beyond the regular languages either fails to be closed under the Boolean operations or lacks virtually all decidability properties.

Context-freeness In Chapter 6, we compare the expressiveness of storage mechanisms with that of pushdown automata. Specifically, we ask which monoids cause valence automata to only accept context-free languages. We characterize those graph products M of monoids for which valence automata over M accept only context-free languages. This means, in particular, that we extend a group-theoretic result of Lohrey and Sénizergues [**LoheySenizergues2007**], which characterizes those graph products of groups where the resulting group is virtually free.

Semilinearity Chapter 7 addresses generalizations of Parikh's Theorem, which states that the Parikh image of each context-free language is semilinear. The first presented result is a characterization of those graph monoids that guarantee semilinear Parikh images. As explained above, this generalizes the semilinearity results for pushdown automata and blind multicounter automata. Moreover, we identify *stacked counters* as expressively complete among those mechanisms with semilinearity. They constitute a new type of storage mechanism that, as shown in later chapters, exhibits a range of properties desirable for analysis. Furthermore, they offer a way to model recursive programs with numeric data types.

Furthermore, it is shown that if G is a torsion group, every language accepted by valence automata over G have a semilinear Parikh image. Moreover, since this semilinearity is not always effective, we characterize those torsion groups for for which semilinear representations are computable.

Silent transitions A *silent transition* is one that reads no input but can manipulate the storage content. For every storage mechanism, it is an important question whether silent transitions are necessary to accept all languages. Indeed, if silent transitions can be eliminated, we can decide the membership status of a given input word by examining a finite number of paths through the automaton. Therefore, in Chapter 8, we ask for which monoids we can avoid silent transitions. We show that among a class of storage mechanisms, stacked counters are those where this is possible. Again, this generalizes the corresponding fact for pushdown automata and blind multicounter automata, which have both been established by Greibach [**Greibach1978**].

Computing downward closures Chapter 9 is concerned with the computation of downward closures. It is well-known that the downward closure, i.e. the set of (not necessarily contiguous) subwords, of every language is regular. Moreover, computing a finite automaton for the downward closure of a given language would make a range of analysis techniques applicable. However, this cannot be done in general. In fact, there are only few known methods for computing downward closures for languages. It is shown here that for all those storage mechanisms that guarantee semilinearity, downward closures can be computed. This generalizes the computability of downward closures for context-free languages, as obtained by **vanLeeuwen1978** [**vanLeeuwen1978**] and **Courcelle1991** [**Courcelle1991**].

The computability result is obtained using the new technique of *Parikh annotations*, for which two other applications are presented.

Non-expressibility Assessing the expressiveness of automata models requires techniques for proving that certain languages cannot be accepted. There-

fore, in Chapter 10, we establish non-expressibility results. The main result is that the hierarchy of languages that is obtained by alternating two transformations of storage mechanisms, *building stacks* and *adding blind counters*, is strict at every level. These are precisely the transformations by which stacked counters are constructed.

This is the fourth application of Parikh annotations.

Arising language classes Finally, in Chapter 11 we examine the limits of valence automata in their ability to model automata with storage. Specifically, we prove general results on the classes of languages that arise from valence automata. The first main result of this section states that every language class defined by valence automata either (i) satisfies a Parikh-type theorem, (ii) contains the partially blind one-counter languages, or (iii) contains the blind one-counter languages.

The second main result establishes that it is not possible in general to add zero-tests. More precisely, there is no transformation that turns a monoid M into a monoid $\tilde{M}$ such that valence automata over $\tilde{M}$ have exactly the capabilities of valence automata over M plus a zero test.

How to read this thesis In order to accommodate readers interested in a specific type of results, this thesis is organized with the aim of minimizing dependencies among chapters.

Chapter 2 introduces almost all concepts that are necessary for understanding each of the remaining chapters. We will sometimes define concepts in the later chapters, but these usually require little explanation and can be looked up quickly. The only exception to this rule are the results on non-expressibility in Chapter 10, which rely on the concept of Parikh annotations. These are introduced and explained in Section 9.2.

Related work Since each mention of related work pertains to a specific chapter, we refer the reader to the conclusion sections therein. In particular, a general overview on related work on valence automata can be found in the conclusion section of the chapter on basic concepts, Section 2.9.

Regarding the introduction, it should be mentioned that the general idea of associating monoids to storage mechanisms as above is old and well-known, see **[Gilman1996]** for a similar approach and references.

Chapter 2

Basic concepts

In this chapter, we settle notation and introduce most of the concepts that are used in the remaining chapters.

Sections 2.1 and 2.2 mostly recall standard terminology on formal languages and monoids. In Section 2.3, we define valence automata and present foundational results. In Section 2.4, we introduce the new concept of graph monoids, which will be used throughout this work as a framework to capture storage mechanisms.

Sections 2.5 and 2.6 are devoted to two operators on language classes that capture the change in expressiveness of two particular constructions of storage mechanisms. These two operators are then used in Section 2.7 to define a hierarchy of language classes that describes the capacity of various types of storage mechanisms in this thesis.

Finally, in Section 2.8, we recall the concept of well-quasi-orderings, which will be used frequently in later chapters.

2.1 Automata and Languages

Sets We denote the set of integers by $\mathbb{Z}$ and the set of non-negative integers by $\mathbb{N}$. To express that the set A is the disjoint union of B and C, we write $A = B \uplus C$. In order to denote inclusion, we write $A \subseteq B$ and for strict inclusion, we write $A \subsetneq B$. The power set of A is denoted by $\mathcal{P}(A)$.

Monoids In this work, a few notions pertaining to formal languages will be defined using monoids. We will therefore begin with some basic concepts for the latter. For further notions concerning monoids, see Section 2.2. For general information about semigroup theory, see **[Grillet1995]**.

A *monoid* is a set M together with a binary associative operation such that M contains a neutral element. The neutral element is called *identity*. Unless the monoid at hand warrants a different notation, we will denote the neutral element by 1 and the product of $x, y \in M$ by xy.

If M and N are monoids, a *morphism* is a map $\varphi\colon M \to N$ with $\varphi(1) = 1$ and $\varphi(xy) = \varphi(x)\varphi(y)$ for any $x, y \in M$. A subset $N \subseteq M$ is a *submonoid* of M if it contains the neutral element of M and satisfies $xy \in N$ whenever $x, y \in N$. For a subset $S \subseteq M$, the *submonoid generated by* S, denoted $\langle S \rangle$, is the smallest

submonoid of M that includes S. It consists of the neutral element of M and all products $s_1 \cdots s_n$ of elements $s_1, \ldots, s_n \in S$. If there is no danger of confusion, we will also write S^* instead of $\langle S \rangle$ and in the case of commutative monoids, we sometimes write $S^\oplus$.

Words and multisets While we define all necessary notions, we assume basic familiarity with formal language theory. For more information, we refer the reader to introductory texts such as [**Kozen1997**, **Berstel1979**, **AutebertBerstelBoasson1997**].

Finite sets of symbols are called *alphabets*. For a set X of symbols, we will write X^* for the set of words over X. The empty word is denoted by $\varepsilon \in X^*$. Together with the concatenation as its operation, X^* is a monoid. For a symbol $x \in X$ and a word $w \in X^*$, let $|w|_x$ be the number of occurrences of x in w. If Y is a subset of X, we write $|w|_Y = \sum_{x \in Y} |w|_x$. By $|w|$, we will refer to the length of w. Given an alphabet X and a monoid M, subsets of X^* and $X^* \times M$ are called *languages* and *transductions*, respectively. For a language $L \subseteq X^*$ and a transduction $T \subseteq X^* \times M$, we define

$$TL = \{m \in M \mid (w, m) \in T \text{ for some } w \in L\}.$$

Given an alphabet X and languages $L, K \subseteq X^*$, the *shuffle product* of L and K is defined as

$$L \sqcup\!\sqcup K = \{u_0 v_1 u_1 \cdots v_n u_n \mid u_0, \ldots, u_n, v_1, \ldots, v_n \in X^*, \\ u_0 \cdots u_n \in L,\ v_1, \ldots, v_n \in K\}.$$

For a subset $Y \subseteq X$, we define the *projection morphism* $\pi_Y \colon X^* \to Y^*$ by setting $\pi_Y(y) = y$ for $y \in Y$ and $\pi_Y(x) = \varepsilon$ for $x \in X \setminus Y$.

Given a set X of symbols, we write $X^\oplus$ for the set of maps $\alpha \colon X \to \mathbb{N}$. The elements of $X^\oplus$ are called *multisets*. By way of pointwise addition, written $\alpha + \beta$, $X^\oplus$ is a commutative monoid. We write 0 for the empty multiset, i.e. the one that maps every $x \in X$ to $0 \in \mathbb{N}$. For $\alpha \in X^\oplus$, let $|\alpha| = \sum_{x \in X} \alpha(x)$.

For each alphabet X, the *Parikh map* is the map $\Psi \colon X^* \to X^\oplus$ defined by $\Psi(w)(x) = |w|_x$ for all $w \in X^*$ and $x \in X$. The map Ψ is lifted to sets in the usual way: $\Psi(L) = \{\Psi(w) \mid w \in L\}$. The set $\Psi(L)$ is then called the *Parikh image* of L. Two languages are said to be *Parikh equivalent* if they have the same Parikh image. For morphisms $\varphi \colon X^\oplus \to Y^\oplus$ and words $w \in X^*$, we sometimes abuse notation and write $\varphi(w)$ instead of $\varphi(\Psi(w))$. Similarly, if $\psi \colon X^* \to Y^*$ is a morphism and $\mu \in X^\oplus$, then $\psi(\mu)$ is defined as $\Psi(\psi(w))$, where $w \in X^*$ satisfies $\Psi(w) = \mu$. Note that this is well-defined.

Rational sets Let M be a monoid. An *automaton over* M consists of a tuple $A = (Q, M, E, q_0, F)$, in which

- Q is a finite set of *states*,
- E is a finite subset of $Q \times M \times Q$ called the set of *edges* or *transitions*,
- $q_0 \in Q$ is the *initial state*, and
- $F \subseteq Q$ is the set of *final states*.

The *step relation* $\to_A$ of A is a binary relation on $Q \times M$, for which we have $(q, m) \to_A (q', m')$ if and only if there is an edge $(q, r, q') \in E$ with $m' = mr$. The set *accepted* by A is then

$$L(A) = \{m \in M \mid (q_0, 1) \to_A^* (f, m) \text{ for some } f \in F\}.$$

A set $R \subseteq M$ is called *rational* if it can be written as $R = L(A)$ for some automaton A over M. The set of rational subsets of M is denoted by $\mathsf{Rat}(M)$. Given two subsets $S, T \subseteq M$, we define $ST = \{st \mid s \in S, t \in T\}$. A classic result by Kleene **[Kleene1956]** states that $\mathsf{Rat}(M)$ is the smallest set of subsets of M that contains every finite subset of M and if $S, T \in \mathsf{Rat}(M)$, then $S^* \in \mathsf{Rat}(M)$ and $ST \in \mathsf{Rat}(M)$. This means, in particular, the operation $(S, T) \mapsto ST$ makes $\mathsf{Rat}(M)$ a monoid itself. Rational languages are also called *regular*. By Reg, we denote the class of regular languages.

Let C be a commutative monoid for which we write the composition additively. For $n \in \mathbb{N}$ and $c \in C$, we use nc to denote $c + \cdots + c$ (n summands). Of course, if we write the composition of C additively, we also write $S + T$ for $\{s + t \mid s \in S, t \in T\}$ and abbreviate $\{s\} + T$ and $S + \{t\}$ as $s + T$ and $S + t$, respectively.

A subset $S \subseteq C$ is called *linear* if there is an element $s \in C$ and a finite set $F \subseteq C$ such that $S = s + \langle F \rangle$. A set $S \subseteq C$ is called *semilinear* if it is a finite union of linear sets. If $S = \bigcup_{i=1}^{n} s_i + \langle F_i \rangle$, then the tuple $(s_1, F_1, \ldots, s_n, F_n)$ is called a *semilinear representation* of S. By $\mathsf{SL}(C)$, we denote the set of semilinear subsets of C. It is well-known that $\mathsf{Rat}(C) = \mathsf{SL}(C)$ for commutative monoids C **[EilenbergSchutzenberger1969]**. We will, however, sometimes still use $\mathsf{SL}(C)$ to make explicit that the sets at hand are semilinear. Clearly, by way of the product $(S, T) \mapsto S + T$, $\mathsf{SL}(C)$ constitutes a commutative monoid. If the set $\Psi(L)$ is semilinear for a language L, we will also call L itself semilinear.

The first-order logic (with equality) over the structure $(\mathbb{N}, +)$ is called *Presburger arithmetic*, its formulae *Presburger formulae*. Here, $+$ is the binary function yielding the sum of its arguments. For a Presburger formula $\varphi(x_1, \ldots, x_n)$ with n free variables $x_1, \ldots, x_n$ we write $\models \varphi(s_1, \ldots, s_n)$ if φ is satisfied in $(\mathbb{N}, +)$ for a variable assignment σ with $\sigma(x_i) = s_i$ for $1 \leqslant i \leqslant n$. For such a formula φ, we say φ *defines* the set $S \subseteq \mathbb{N}^n$ if for each $(s_1, \ldots, s_n) \in \mathbb{N}^n$, we have $\models \varphi(s_1, \ldots, s_n)$ if and only if $(s_1, \ldots, s_n) \in S$. A subset $S \subseteq \mathbb{N}^n$ is said to be *Presburger definable* if it is defined by some Presburger formula.

A classic result by Ginsburg and Spanier **[GinsburgSpanier1966]** states that the Presburger definable subsets of $\mathbb{N}^n$ are precisely the semilinear ones. Moreover, this equivalence is effective, meaning that given a Presburger formula, one can effectively determine a semilinear representation for the set it defines. If X is an alphabet, then a linear order on X induces an isomorphism $X^{\oplus} \cong \mathbb{N}^n$, where $n = |X|$, by way of which the notion of Presburger definability carries over to subsets of $X^{\oplus}$. It is clearly independent of the chosen linear order.

Language classes and closure properties This work studies, among other aspects, the expressive power of valence automata, which is measured by the languages they accept. In order to discuss the resulting language classes, we need a few fundamental notions. A *rational transduction* is a rational subset T of the monoid $X^* \times Y^*$ for alphabets X and Y. If $A = (Q, X^* \times Y^*, E, q_0, F)$ is an automaton over $X^* \times Y^*$, we also write $A = (Q, X, Y, E, q_0, F)$ and instead of $L(A)$, we also

use the notation $T(A)$. For rational transductions $T \subseteq X^* \times Y^*$ and $U \subseteq Y^* \times Z^*$, we define

$$T \circ U = \{(u, w) \in X^* \times Z^* \mid (u, v) \in T,\ (v, w) \in U \text{ for some } v \in Y^*\}.$$

A classic result by Elgot and Mezei **[ElgotMezei1965]** states that $U \circ T$ is again a rational transduction.

A *substitution* is a map $\sigma\colon X \to \mathcal{P}(Y^*)$, where X and Y are alphabets. Given $L \subseteq X^*$, we write $\sigma(L)$ for the set of all words $v_1 \cdots v_n$, where $v_i \in \sigma(x_i)$, $1 \leqslant i \leqslant n$, for $x_1 \cdots x_n \in L$ and $x_1, \ldots, x_n \in X$. If $\sigma(x) \subseteq Y$ for each $x \in X$, we call σ a *letter substitution*.

A *language class* is a set of languages that is closed under isomorphism and contains at least one non-empty member. A language class $\mathcal{C}$ is called a *full trio* if it is closed under rational transductions, that is, for each $L \in \mathcal{C}$, $L \subseteq X^*$, and each rational transduction $T \subseteq X^* \times Y^*$, we have $TL \in \mathcal{C}$. It follows from the definition that every full trio includes the regular languages. Moreover, it is a simple exercise to show that for $L \in \mathcal{C}$, $L \subseteq X^*$, regular languages $R \subseteq X^*$ and morphisms $h\colon X^* \to Y^*$, and $g\colon Y^* \to X^*$, one has

$$L \cap R \in \mathcal{C}, \qquad L \sqcup\!\sqcup R \in \mathcal{C}, \qquad h(L) \in \mathcal{C}, \qquad g^{-1}(L) \in \mathcal{C}. \tag{2.1}$$

We denote the smallest full trio containing L by $\mathcal{T}(L)$. Since $T \circ U$ is rational when T and U are rational transductions, $\mathcal{T}(L)$ consists of all languages of the form TL, where T is a rational transduction; unless $L = \emptyset$, in which case $\mathcal{T}(L)$ coincides with the regular languages. If a language class is of the form $\mathcal{T}(L)$ (i.e. generated by one language), it is said to be a *principal full trio*.

A full trio is called a *full semi-AFL*[1] if it is also closed under finite unions, i.e. if $K, L \in \mathcal{C}$, then $K \cup L \in \mathcal{C}$. Furthermore, a semi-AFL is said to be a *full AFL* if for each $L \in \mathcal{C}$, we have $L^* \in \mathcal{C}$. For more information on these concepts, see **[Berstel1979]**. A language class is said to be *semilinear* if each of its languages is semilinear.

Certain constructions in this work rely on a somewhat unusual combination of closure properties, which makes it convenient to give it a name. First, we say that a language class $\mathcal{C}$ is *Presburger closed*[2] if for each language $L \in \mathcal{C}$, $L \subseteq X^*$, and each semilinear set $S \subseteq X^{\oplus}$, we have $L \cap \Psi^{-1}(S) \in \mathcal{C}$. Moreover, a rational transduction $T \subseteq X^* \times Y^*$ is called *locally finite* if the set TL is finite for each finite $L \subseteq X^*$. Second, we say that a language class is a *full semi-trio* if it is closed under locally finite transductions. Just as full trios always include the regular languages, full semi-trios always include the finite languages. Note also that if we require g to be non-erasing, i.e. $g(y) \neq \varepsilon$ for $y \in Y$, then (2.1) still holds for $L \in \mathcal{C}$, where $\mathcal{C}$ is a full semi-trio. An example of a full semi-trio that is also Presburger closed is the class of finite languages. See Section 9.2 for the aforementioned constructions that require the input languages to belong to a Presburger closed full semi-trio.

A note on effectiveness We will often say that a language class is *effectively* closed under some operation. By this we mean that given representations of languages in the class, one can effectively determine a representation of the resulting

[1] In the terms semi-AFL and AFL, the abbreviation "AFL" stands for Abstract Family of Languages **[Berstel1979]**.

[2] The name stems from the fact that the semilinear sets are precisely those defined by Presburger formulae; see p. 9.

language. For example, a language class $\mathcal{C}$ is *effectively closed under rational transductions* if there is an algorithm that, given a representation of some language $L \in \mathcal{C}$ and a rational transduction T, computes a representation of the language $TL \in \mathcal{C}$. It is *effectively Presburger closed* if, given a representation of a language $L \in \mathcal{C}$ and a semilinear representation of a set S, one can compute a representation of $L \cap \Psi^{-1}(S)$, etc. This convention carries over to types of language classes: An *effective full trio* is a language class that is effectively closed under rational transductions, etc.

It will always be clear from the definition of a language class how its members are represented. For example, a blind multicounter language is described by a blind multicounter automaton. A language in $\mathsf{Alg}(\mathcal{C})$ (see Section 2.6 for a definition) is represented by a $\mathcal{C}$-grammar together with descriptions of its right-hand sides, which will again be clear how to represent.

Pushdown automata and context-free languages A *pushdown automaton* is a tuple $A = (Q, X, Y, E, q_0, F)$, where Q is a finite set of *states*, X and Y are alphabets, $E \subseteq Q \times X^* \times Y^* \times Y^* \times Q$ is a finite set of *edges* (or *transitions*), $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *final states*. The alphabet X is called its *input alphabet* and Y its *stack alphabet*. A *configuration* of A is a triple (q, u, v), where $q \in Q$, $u \in X^*$, and $v \in Y^*$. For such a pushdown automaton A and configurations (q, u, v) and (q', u', v'), we write

$$(q, u, v) \to_A (q', u', v') \qquad \begin{array}{l} \text{if } u' = ux,\ v = wy_1, \text{ and } v' = wy_2 \\ \text{for some } w \in Y^* \text{ and } (q, x, y_1, y_2, q') \in E. \end{array}$$

The language *accepted by* A is then

$$L(A) = \{w \in X^* \mid (q_0, \varepsilon, \varepsilon) \to_A^* (f, w, \varepsilon) \text{ for some } f \in F\}.$$

Note that this definition deviates slightly from the usual textbook definition [**Berstel1979**]. In the latter, every edge (p, x, y_1, y_2, q) has $|y_1| = 1$. This means in particular that in the initial configuration, the stack has to consist of some designated bottom symbol. Moreover, in our definition, a computation has to end in a configuration where both a final state is reached and the stack is empty. The textbook definition usually has two variants: One accepts on final state and one accepts with an empty stack. In terms of the class of accepted languages, it is easy to see that all three definitions are equivalent.

The languages accepted by pushdown automata are called *context-free* and we use CF to denote the class of context-free languages. An important tool for proving that a certain language is not context-free is the Iteration Lemma by Ogden [**Ogden1968**].

Theorem 2.1.1 (Ogden [**Ogden1968**])**.** *For each context-free language L, there is an integer m such that for any word $z \in L$ and any choice of at least m distinct marked positions in z, there is a decomposition $z = uvwxy$ such that:*

1. *w contains at least one marked position.*
2. *Either u and v both contain marked positions, or x and y both contain marked positions.*
3. *vwx contains at most m marked positions.*

4. $uv^i wx^i y \in L$ *for every* $i \geqslant 0$.

Another tool to examine context-freeness of languages is Parikh's theorem. It is useful for proving that particular languages are not context-free, but also for decision problems involving context-free languages. For some examples of its numerous applications, see Chapter 7.

Theorem 2.1.2 (Parikh **[Parikh1966]**). *For each context-free language* L, $\Psi(L)$ *is effectively semilinear.*

An important characterization of the context-free languages is the theorem of Chomsky and Schützenberger **[ChomskySchutzenberger1963]** (see also **[Berstel1979]**). In order to formulate it, we have to define Dyck languages. For each $n \in \mathbb{N}$, let X_n be the alphabet $\{a_i, \bar{a}_i \mid 1 \leqslant i \leqslant n\}$. The *Dyck language (over* n *pairs of parentheses)* is the smallest language $D_n \subseteq X_n^*$ containing ε such that for each $uv \in D_n$, D_n also contains $ua_i\bar{a}_i v$ and $u\bar{a}_i a_i v$. The *semi-Dyck language (over* n *pairs of parentheses)* is the smallest language $D'_n \subseteq X_n^*$ that contains ε and for each $uv \in D'_n$, D'_n also contains $ua_i\bar{a}_i v$.

Theorem 2.1.3 (Chomsky-Schützenberger **[ChomskySchutzenberger1963]**). $\mathsf{CF} = \mathcal{T}(D_2) = \mathcal{T}(D'_2)$.

Counter automata Let $n \in \mathbb{N}$ and S be a subset of $\mathbb{Z}^n$. An *S-restricted counter automaton* is a tuple $A = (Q, X, E, q_0, F)$, where Q is a finite set of *states*, X is an alphabet, E is a finite subset of $Q \times X^* \times \mathbb{Z}^n \times Q$ for some $n \in \mathbb{N}$, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *final states*. A *configuration* of A is a triple (q, u, μ) with $q \in Q$, $u \in X^*$, and $\mu \in S$. For such an automaton and configurations (q, u, μ) and (q', u', μ'), we write

$$(q, u, \mu) \to_A (q', u', \mu') \quad \text{if } u' = ux \text{ and } \mu' = \mu + \kappa \text{ for some } (q, x, \kappa, q') \in E.$$

The language *accepted by* A is then

$$L(A) = \{w \in X^* \mid (q_0, \varepsilon, 0) \to_A^* (f, w, 0) \text{ for some } f \in F\}.$$

A $\mathbb{Z}^n$-restricted counter automaton is called a *blind (multi)counter automaton* and an $\mathbb{N}^n$-restricted counter automaton is called a *partially blind (multi)counter automaton*. While we will not formally introduce the model of Petri nets in this work, let us mention that, when operated with labels and final markings **[Jantzen1979]**, they are essentially equivalent[3] to partially blind multicounter automata.

2.2 Monoids

Since in this work, we represent storage mechanisms by monoids, they constitute a central concept. This section recalls basic notions.

Let M be a monoid. A *congruence* is an equivalence relation $\equiv$ on M such that $x \equiv y$ implies $uxv \equiv uyv$ for every $u, v \in M$. Given a congruence $\equiv$, we can define the *quotient monoid* $M/\equiv$, whose elements are the equivalence classes with respect to $\equiv$. The operation on $M/\equiv$ is given by $[x]_\equiv [y]_\equiv = [xy]_\equiv$. The fact

[3] Strictly speaking, Petri nets do not have states, but this is usually without consequence because they can be compensated with additional places. This does, however, mean that partially blind multicounter automata with n counters correspond to Petri nets with n *unbounded* places.

that $\equiv$ is a congruence guarantees that this operation is well-defined. The *trivial monoid* is the monoid that contains just one element and is denoted **1**. A bijective morphism is called *isomorphism* and we say that two monoids are *isomorphic* if there is an isomorphism between them.

If $M_1, \ldots, M_n$ are monoids, then their *direct product* consists of the Cartesian product $M_1 \times \cdots \times M_n$ and the operation that applies M_i's product in the i-th component. We write M^n for the direct product $M \times \cdots \times M$ (n factors).

In each monoid M, we have the subsets

$$\begin{aligned} R_1(M) &= \{x \in M \mid xy = 1 \text{ for some } y \in M\}, \\ L_1(M) &= \{x \in M \mid yx = 1 \text{ for some } y \in M\}, \\ H_1(M) &= \{x \in M \mid xy = yx = 1 \text{ for some } y \in M\}, \\ J_1(M) &= \{x \in M \mid yxz = 1 \text{ for some } y, z \in M\}. \end{aligned}$$

It should be noted that $R_1(M)$, $L_1(M)$, and $J_1(M)$ are the $\mathcal{R}$-, $\mathcal{L}$-, $\mathcal{H}$- and $\mathcal{J}$-class, respectively, of the identity in M with respect to the well-known Green's relations $\mathcal{R}$, $\mathcal{L}$, $\mathcal{H}$, and $\mathcal{J}$ **[Grillet1995]**. Green's relations are an important concept in the theory of semigroups and monoids. For our purposes, however, it suffices to consider the sets $R_1(M)$, $L_1(M)$, $H_1(M)$, and $J_1(M)$. Observe that $R_1(M)$, $L_1(M)$, and $H_1(M)$ are each a submonoid of M.

The elements of $R_1(M)$, $L_1(M)$, and $H_1(M)$ are also called *right-invertible*, *left-invertible*, and *invertible*, respectively. If every element of M is invertible, then M is a *group*. A *subgroup* of a monoid M is a submonoid[4] that is a group. Note that $H_1(M)$ is always a subgroup of M.

Presentations and free products One way to describe a monoid is to define its set of elements and specify the product of any given pair of them. Another approach is the following. Instead of explicitly providing the product for any pair, one writes down a set of generators and indicates what equations are to hold among their products. While the direct product of monoids is simple from the former perspective, the free product is very natural from the second. A description of a monoid by generators and equations is called a presentation. Let A be a (not necessarily finite) set of symbols and $R \subseteq A^* \times A^*$. The pair (A, R) is called a *(monoid) presentation*. The smallest congruence of A^* containing R is denoted by $\equiv_R$ and we will write $[w]_R$ for the congruence class of $w \in A^*$. The *monoid presented by* (A, R) is defined as $A^*/\equiv_R$. Hence, two words $u, v \in A^*$ represent the same element of the monoid if u can be transformed into v by repeatedly replacing factors x by y for $(x, y) \in R$ or $(y, x) \in R$.

Note that since we did not impose a finiteness restriction on A, every monoid M has a presentation (up to isomorphism): Take a set of symbols A in bijection with M and let $\varphi\colon A^* \to M$ be the morphism extending this bijection. With $R = \{(u, v) \in A^* \times A^* \mid \varphi(u) = \varphi(v)\}$, the presentation (A, R) presents a monoid isomorphic to M.

Furthermore, for monoids M_0, M_1 we can find presentations (A_0, R_0) and (A_1, R_1) such that $A_0 \cap A_1 = \emptyset$. We define the *free product* $M_0 * M_1$ to be presented by $(A_0 \cup A_1, R_0 \cup R_1)$. Note that this definition of the free product depends on the chosen presentation. However, it is easy to see that $M_0 * M_1$ is

[4] Note that here, a subgroup being a submonoid implies that the identity of the subgroup coincides with the identity of M. In the literature on semigroup theory, subgroups are usually only required to be subsemigroups, meaning that the subgroup's identity can be any idempotent.

well-defined up to isomorphism: If (A_i, R_i) and (A'_i, R'_i) present the same monoid for $i = 0, 1$, then there are morphisms $\varphi_i \colon A_i^* \to A_i'^*$ such that $u \equiv_{R_i} v$ if and only if $\varphi_i(u) \equiv_{R'_i} \varphi_i(v)$ for $i = 0, 1$ and for each $w \in A_i'^*$, $i \in \{0, 1\}$, there is a $v \in A_i^*$ with $\varphi(v) \equiv_{R'_i} w$. If $\varphi \colon (A_0 \cup A_1)^* \to (A'_0 \cup A'_1)^*$ is the morphism extending φ_0 and φ_1, then the induced map

$$\begin{aligned}(A_0 \cup A_1)^*/{\equiv_{R_0 \cup R_1}} &\longrightarrow (A'_0 \cup A'_1)^*/{\equiv_{R'_0 \cup R'_1}} \\ [w]_{R_0 \cup R_1} &\longmapsto [\varphi(w)]_{R'_0 \cup R'_1}\end{aligned}$$

is well-defined and an isomorphism.

Since A_0 and A_1 are disjoint, the morphisms defined by $[w]_{R_i} \mapsto [w]_{R_0 \cup R_1}$, $w \in A_i^*$ for $i = 0, 1$, are injective. By way of these, we will regard M_0 and M_1 as subsets of $M_0 * M_1$. Since every word $w \in (A_0 \cup A_1)^*$ can be written as $w = u_0 v_1 u_1 \cdots v_n u_n$ with $u_i \in A_0^*$, $0 \leqslant i \leqslant n$, and $v_i \in A_1^*$, $1 \leqslant i \leqslant n$, we can write every element of $M_0 * M_1$ as a product $x_0 y_1 x_1 \cdots y_n x_n$ with $x_i \in M_0$, $0 \leqslant i \leqslant n$, and $y_i \in M_1$, $1 \leqslant i \leqslant n$. These decompositions are not always unique: If, for example, $x_j = 1$, then

$$\begin{aligned}x_0 y_1 x_1 \cdots y_n x_n &= x_0 y_1 x_1 \cdots y_{j-1} x_{j-1} y_j x_j y_{j+1} x_{j+1} \cdots y_n x_n \\ &= x_0 y_1 x_1 \cdots y_{j-1} x_{j-1} \underbrace{(y_j y_{j+1})}_{\in M_1} x_{j+1} \cdots y_n x_n.\end{aligned}$$

However, this is the only situation in which the decomposition is not unique. With the requirement $x_i \neq 1$ for $0 \leqslant i \leqslant n$ and $y_i \neq 1$ for $1 \leqslant i \leqslant n$, the elements $x_0, \ldots, x_n$ and $y_1, \ldots, y_n$ are uniquely determined.

The definition of the free product also implies that for every pair of morphisms $\psi_i \colon M_i \to N$, $i = 0, 1$, for some monoid N, there is a unique morphism $\psi \colon M_0 * M_1 \to N$ extending ψ_0 and ψ_1, i.e. ψ restricted to M_i agrees with ψ_i for $i = 0, 1$. Furthermore, it is immediate from the definition that the free product is associative, i.e. $(M * N) * P \cong M * (N * P)$ for monoids M, N, P. Therefore, we may define the n-fold free product of M by $M^{(n)} = M * \cdots * M$ (n factors).

The bicyclic monoid One of the central monoids in this work is the bicyclic monoid. We define it by a presentation: Let $X = \{x, \bar{x}\}$ and $R = \{(x\bar{x}, \varepsilon)\}$. Then we call the monoid presented by (X, R) the *bicyclic monoid* and denote it by $\mathbb{B}$. The elements $[x]_R$ and $[\bar{x}]_R$ of $\mathbb{B}$ are also denoted a and $\bar{a}$, respectively. They are called the *positive* and *negative generator*, respectively. Observe that every element of $\mathbb{B}$ has a unique representation $\bar{a}^m a^n$ for $m, n \in \mathbb{N}$.

2.3 Valence automata

The aim of this work is to generalize insights about automata with storage. More specifically, we want to generalize results about concrete storage mechanisms and then, for some larger class of mechanisms, characterize those mechanisms for which the result still holds and for which it fails.

In order to discuss classes of storage mechanisms, we need a formal model of automata with storage in which the storage mechanism is given by some parameter. This means by choosing suitable values for the parameter, one can instantiate

concrete automata models. With such a model, it is possible to ask: For which parameter values does the result hold?

A model that turns out to be suitable for this endeavor is the model of valence automata. Such an automaton consists of a finite state control and a storage mechanism that is defined by a (possibly infinite) monoid. Here, both the current content of the storage and the operations thereon are represented as elements of this monoid: In the beginning, the storage content is the neutral element of the monoid and in each step, the automaton operates on the storage by multiplying an element indicated on the used transition.

Valence automata Let us define the model formally. Let M be a monoid. A *valence automaton over* M is a tuple $A = (Q, X, M, E, q_0, F)$, where

- Q is a finite set of *states*,
- X is an alphabet, called its *input alphabet*,
- M is a monoid,
- $E \subseteq Q \times X^* \times M \times Q$ is a finite set of *edges* or *transitions*,
- $q_0 \in Q$ is its *initial state*, and
- $F \subseteq Q$ is its set of *final states*.

A *configuration* is a triple $(q, u, x) \in Q \times X^* \times M$. For configurations (q, u, x) and (q', u', x'), let

$$(q, u, x) \to_A (q', u', x') \qquad \text{if } u' = uw \text{ and } x' = xz \text{ for some } (q, w, z, q') \in E.$$

The term 'valence automaton' stems from the analogous concept of valence grammars, which were introduced by **Paun1980** [**Paun1980**]. Details on the latter can be found in Chapter 3. However, the concept of valence automata had been known and studied before the work [**Paun1980**]. Some authors also say M-automata instead of valence automata over M. For information on related work on valence automata and other unifying models, the reader is referred to Section 2.9.

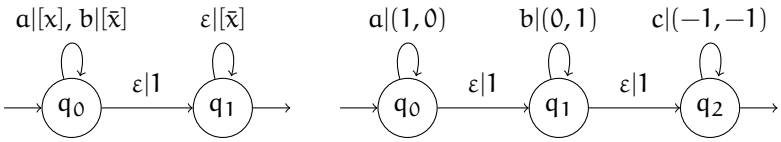
In this work, we measure the expressive power of valence automata by the class of languages they can accept. The *language accepted by* A is defined as

$$L(A) = \{w \in X^* \mid (q_0, \varepsilon, 1) \to_A^* (f, w, 1) \text{ for some } f \in F\}.$$

In other words, the automaton accepts all words that label paths from an initial state to a final state such that the product of the monoid elements is the identity.

The class of languages accepted by valence automata over M is denoted by $\mathsf{VA}(M)$. Sometimes, we will consider all valence automata over monoids from a class $\mathcal{M}$. We will then write $\mathsf{VA}(\mathcal{M})$ for the class of languages accepted by some valence automaton over M with $M \in \mathcal{M}$.

Example 2.3.1. *Consider the valence automata* A *and* B *in Figs. 2.1a and 2.1b. The former is a valence automaton over* $\mathbb{B}$ *and the latter is one over* $\mathbb{Z} \times \mathbb{Z}$*. We denote a transition* (p, w, m, q) *by drawing an edge labeled* $w|m$ *from* p *to* q*. Initial states*



(a) Valence automaton A over $\mathbb{B}$ (b) Valence automaton B over $\mathbb{Z}^2$

Figure 2.1: Examples of valence automata

and final states are marked by an incoming and an outgoing arrow, respectively. The languages accepted by our example automata are

$$L(A) = \{w \in \{a,b\}^* \mid |u|_a \geqslant |u|_b \text{ for each prefix } u \text{ of } w\},$$
$$L(B) = \{a^n b^n c^n \mid n \in \mathbb{N}\}.$$

Before we see in Section 2.4 how to realize concrete storage mechanisms with monoids, we presents here some basic observations concerning valence automata with respect to expressive power and decidability. Among other applications, these will be helpful for understanding how the monoids in Section 2.4 correspond to storage mechanisms.

For the description of the languages accepted by valence automata over a certain monoid, the identity languages play an important role. An *identity language of* M is a language of the form $\varphi^{-1}(1)$, where $\varphi\colon X^* \to M$ is a morphism and X is an alphabet.

Example 2.3.2. *Consider the additive group $\mathbb{Z}$ and the morphism $\varphi\colon X^* \to \mathbb{Z}$ with $X = \{a,b\}$ and $\varphi(a) = 1$ and $\varphi(b) = -1$. Here, of course, 1 is the usual integer and not the neutral element of $\mathbb{Z}$. Then the identity language of $\mathbb{Z}$ with respect to φ is $\{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$.*

Suppose $\varphi_i\colon X_i^* \to M_i$ is a morphism for $i = 0,1$ such that $X_0 \cap X_1 = \emptyset$. Then the language $\varphi_0^{-1}(1) \sqcup\!\sqcup \varphi_1^{-1}(1)$ is an identity language of the monoid $M_0 \times M_1$.

The class of languages accepted by valence automata over M can be characterized in terms of the identity languages of M. This has been observed, for example, by **GilmanShapiro1998 [GilmanShapiro1998]** and **Kambites2009 [Kambites2009]**.

Theorem 2.3.3. *Let M be a monoid. The following are equivalent.*

1. $L \in \mathsf{VA}(M)$
2. L *is a rational transduction of some identity language of* M.

In other words, $\mathsf{VA}(M)$ is the smallest full trio containing all identity languages of M.

Proof. Let $A = (Q, X, M, E, q_0, F)$ be a valence automaton over M. Let $S \subseteq M$ be the finite set of those $m \in M$ for which there is a transition (p, w, m, q) in E. Moreover, let Y be an alphabet in bijection with S and let $\varphi\colon Y^* \to M$ be the morphism extending this bijection. Then the transducer $B = (Q, Y, X, E', q_0, F)$ with

$$E' = \{(p, \varphi^{-1}(m), w, q) \mid (p, w, m, q) \in E\}$$

is easily seen to satisfy $L(A) = T(B)\varphi^{-1}(1)$ (note that $T(B) \subseteq Y^* \times X^*$).

On the other hand, if $\varphi\colon Y^* \to M$ is a morphism and $A = (Q, Y, X, E, q_0, F)$ is a transducer, then letting

$$E' = \{(p, w, \varphi(v), q) \mid (p, v, w, q) \in E\}$$

defines a valence automaton $B = (Q, X, M, E', q_0, F)$ with $L(B) = T(A)\varphi^{-1}(1)$. □

Since the class of rational transductions is closed under composition, the foregoing fact implies that $\mathsf{VA}(M)$ is a full trio for each monoid M. It is also easy to see that each $\mathsf{VA}(M)$ is closed under union. This implies the following, which has also been mentioned by Fernau and Stiebe [**FernauStiebe2002a**].

Corollary 2.3.4. *For each monoid M, $\mathsf{VA}(M)$ is a full semi-AFL.*

Using a finitely generated monoid as a storage monoid means that there is a fixed finite set S that generates M. We can then write every monoid element on a transition as a product of elements of S. Intuitively, this means there is a globally fixed number of operations one can apply to the storage. This is the case, for example, in the class of pushdown automata with a fixed pushdown alphabet; or in the class of multicounter automata with a fixed number of counters. Note that in the case of pushdown automata, confining oneself to two pushdown symbols does not affect the class of accepted languages. For blind multicounter automata, on the other hand, the number of counters induces a strict hierarchy of languages (Theorem 10.1.1).

If we are interested in whether a fixed number of operations suffices in this sense, the following provides a necessary and sufficient condition for the resulting language class. Interestingly, this means whether "a finite subset of operations suffices" does not depend on the monoid M or a chosen generating set for M: It only depends on the structure of the resulting language class.

Corollary 2.3.5. *For each monoid M, the following conditions are equivalent:*

1. *There is a finitely generated submonoid N of M with $\mathsf{VA}(N) = \mathsf{VA}(M)$.*
2. *$\mathsf{VA}(M)$ is a principal full trio.*

Proof. For the direction "1 ⇒ 2", we show that $\mathsf{VA}(N)$ is a principal full trio if N is finitely generated. In this case, there is a surjective morphism $\varphi\colon X^* \to N$. We claim that $\mathsf{VA}(N)$ is generated, as a full trio, by the identity language $L = \varphi^{-1}(1)$. By Theorem 2.3.3, it suffices to show that every identity language of N belongs to $\mathcal{T}(L)$. Suppose $K = \psi^{-1}(1)$ is another identity language for $\psi\colon Y^* \to N$. Since φ is surjective, there is for each $y \in Y$ a word $w_y \in X^*$ with $\varphi(w_y) = \psi(y)$. Hence, if $\psi'\colon Y^* \to X^*$ is the morphism with $y \mapsto w_y$ for $y \in Y$, then the language

$$K = \psi^{-1}(1) = \psi'^{-1}(\varphi^{-1}(1)) = \psi'^{-1}(L)$$

clearly belongs to $\mathcal{T}(L)$.

Let us prove "2 ⇒ 1". Suppose $\mathsf{VA}(M)$ is a principal full trio. Then there is a language $L \in \mathsf{VA}(M)$ such that every $K \in \mathsf{VA}(M)$ can be written as $K = TL$ for some rational transduction T. In the valence automaton A over M for L, only a finite set S of elements of M occurs on the transitions. This means, for the finitely generated submonoid $N = \langle S \rangle$, we have $L \in \mathsf{VA}(N)$. Since $\mathsf{VA}(N)$ is a full trio, we have $\mathsf{VA}(M) \subseteq \mathsf{VA}(N)$ and thus $\mathsf{VA}(N) = \mathsf{VA}(M)$. □

Suppose the two monoids M_0 and M_1 each realize a particular storage mechanism. Then it is easy to see that the monoid $M_0 \times M_1$ realizes the mechanism in which one has access to both factors independently and simultaneously. The next theorem describes the effect of such a construction on the expressive power. It appeared first in [**Kambites2009**] and can be shown using a simple product construction.

Theorem 2.3.6 (Kambites2009 [Kambites2009]). *Let $M_1, \ldots, M_n$ be monoids. Then the language class $\mathsf{VA}(M_1 \times \cdots \times M_n)$ consists of precisely those languages of the form*

$$h(L_1 \cap \cdots \cap L_n),$$

where $L_i \in \mathsf{VA}(M_i)$ for $1 \leqslant i \leqslant n$ and h is a morphism.

An immediate consequence of the previous theorem is the following.

Corollary 2.3.7. *If $\mathsf{VA}(N_i) \subseteq \mathsf{VA}(M_i)$ for $i = 0, 1$, then*

$$\mathsf{VA}(N_0 \times N_1) \subseteq \mathsf{VA}(M_0 \times M_1).$$

The next lemma is also a consequence of Theorem 2.3.6 and will mostly be used in contraposition: If we already know that for some n, the class $\mathsf{VA}(M)$ differs from $\mathsf{VA}(M \times N^n)$, it allows us to conclude $\mathsf{VA}(M) \neq \mathsf{VA}(M \times N)$.

Lemma 2.3.8. *Let M and N be monoids. If $\mathsf{VA}(M \times N) = \mathsf{VA}(M)$, then for every $n \in \mathbb{N}$, we have $\mathsf{VA}(M \times N^n) = \mathsf{VA}(M)$.*

Proof. Suppose $\mathsf{VA}(M \times N) = \mathsf{VA}(M)$ and $L \in \mathsf{VA}(M \times N^n)$. By Theorem 2.3.6, this means $L = h(K \cap L_1 \cap \cdots \cap L_n)$ for some $K \in \mathsf{VA}(M)$, $L_i \in \mathsf{VA}(N)$ for $1 \leqslant i \leqslant n$, and a morphism h. By induction on i, it follows that the intersection $K \cap L_1 \cap \cdots \cap L_i$ belongs to $\mathsf{VA}(M)$: For $i = 0$ this is trivial and if it holds for i, then

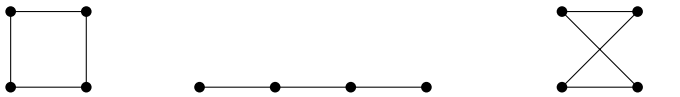
$$K \cap L_1 \cap \cdots \cap L_{i+1} = (K \cap L_1 \cap \cdots \cap L_i) \cap L_{i+1} \in \mathsf{VA}(M \times N) \subseteq \mathsf{VA}(M)$$

by Theorem 2.3.6. In particular, $K \cap L_1 \cap \cdots \cap L_n \in \mathsf{VA}(M)$ and hence L belongs to $\mathsf{VA}(M)$. □

We close this section by presenting two transformations of monoids that can be easily interpreted in terms of the storage mechanisms they yield.

Direct products and independent mechanisms Suppose a valence automaton A has $M_0 \times M_1$ as its storage monoid. This means, every transition of A holds an element of M_0 and an element of M_1 and applies them to the respective storage components. In the end, it accepts if both components contain the identity.

This means, *taking direct products corresponds to using both mechanisms independently*. For example, since $\mathbb{Z}$ is essentially one blind counter, using the monoid $M \times \mathbb{Z}$ instead of M means we *add a blind counter*.



(a) C_4 (b) P_4 (c) C_4, drawn differently

Figure 2.2: Graphs C_4 and P_4.

Free products and building stacks Let us consider what happens when we go from M to $\mathbb{B} * M$. By the definition of the free product and the bicyclic monoid, it is easy to see that if $m \in M$, $m \neq 1$, then $a m \bar{a}$ cannot occur in a product that yields the identity: Any word obtained by applying the equations will contain the symbols a and $\bar{a}$ and a word representing $m \neq 1$ in between them. Similarly, an element $m\bar{a}x$ with $m \in M$ and $x \in \mathbb{B} * M$ is not right-invertible. This implies that every right-invertible element of $\mathbb{B} * M$ is of the form $m_0 a m_1 \cdots a m_n$, in which $m_0, \ldots, m_n \in M$ are uniquely determined.

We interpret this as a *stack of entries in* M, where a is the separator between entries. Observe that multiplying $\bar{a}$ yields a right-invertible element if and only if $m_n = 1$ and if that is the case, we arrive at $m_0 a m_1 \cdots a m_{n-1}$. Hence, $\bar{a}$ acts as a *pop* operation. Furthermore, multiplying a is a *push* operation that starts a new entry on the top. Moreover, multiplying an element of M manipulates the topmost entry. Finally, $m_0 a m_1 \cdots a m_n$ is the identity of $\mathbb{B} * M$ if and only if $n = 0$ and $m_0 = m_1 = 1$, meaning that the stack is empty.

Therefore, taking the free product with $\mathbb{B}$ corresponds to *building stacks*. In particular, in the case $M = \mathbf{1}$, we build stacks of trivial elements and hence obtain a *partially blind counter*. If $M = \mathbb{B}$, then we have a stack of partially blind counters, which is precisely a *pushdown over two symbols*.

2.4 Graph monoids

This section introduces a class of monoids that accommodates a variety of storage mechanisms. We call these monoids *graph monoids*[5].

Aside from realizing a number of storage mechanisms, they have the advantage that they are closely related to Mazurkiewicz traces **[DiekertRozenberg1995]** and that they generalize graph groups (which are also known under the name right-angled Artin groups) **[Charney2007]**. The connections to Mazurkiewicz traces and graph groups allow us to use ideas and results from these areas to understand valence automata over graph monoids. On the other hand, the language- and automata-theoretic perspective sometimes simplifies difficult proofs about graph groups (see, for example, Theorem 2.6.3).

Graphs Graph monoids are defined by graphs. A *graph* is a pair $\Gamma = (V, E)$ where V is a finite set and E is a subset of $\{S \subseteq V \mid 1 \leqslant |S| \leqslant 2\}$. The elements

[5]They are not to be confused with the related, but different concept of *trace monoids* **[DiekertRozenberg1995]**, i.e. monoids of Mazurkiewicz traces, which some authors also call graph monoids.

of V are called *vertices* and those of E are called *edges*. This means, in our definition of graphs, edges are undirected, but we allow loops. We call Γ *simple* if it has no loops, i.e. $|S| = 2$ for every $S \in E$. Vertices $v, w \in V$ are *adjacent* if $\{v, w\} \in E$. If $\{v\} \in E$ for some $v \in V$, then v is called a *looped* vertex, otherwise it is *unlooped*. Given a graph $\Gamma = (V, E)$, we write Γ^- for its *underlying simple graph*, that is, the graph $\Gamma' = (V, E')$, where $E' = E \cap \{S \subseteq V \mid |S| = 2\}$. We call graphs $\Gamma_0 = (V_0, E_0)$ and $\Gamma_1 = (V_1, E_1)$ *isomorphic* if there is a bijection $\varphi\colon V_0 \to V_1$ such that $\{v, w\} \in E_0$ if and only if $\{\varphi(v), \varphi(w)\} \in E_1$.

A *subgraph* of Γ is a graph (V', E') with $V' \subseteq V$ and $E' \subseteq E$. Such a subgraph is called *induced (by V')* if $E' = \{S \in E \mid S \subseteq V'\}$, i.e. E' contains all edges from E whose incident vertices are in V'. By $\Gamma \setminus v$, for $v \in V$, we denote the subgraph of Γ induced by $V \setminus \{v\}$. For a vertex $v \in V$, the elements of the set $N(v) = \{w \in V \mid \{v, w\} \in E\}$ are called *neighbors* of v.

A *looped clique* is a graph in which $E = \{S \subseteq V \mid 1 \leqslant |S| \leqslant 2\}$. Moreover, a *clique* is a loop-free graph in which any two distinct vertices are adjacent. Finally, an *anti-clique* is a graph with $E = \emptyset$. A *simple path (of length n)* is a sequence $x_1, \ldots, x_n$ of pairwise distinct vertices such that $\{x_i, x_{i+1}\} \in E$ for $1 \leqslant i < n$. If, in addition, we have $\{x_n, x_1\} \in E$, it is called a *cycle (of length n)*. By C_4, we denote the graph with four vertices that constitute a cycle such that C_4 has a minimal set of edges. Similarly P_4 is the graph with four vertices that constitute a simple path such that P_4 has a minimal set of edges (see Figs. 2.2a and 2.2b).

In slight abuse of terminology, we say the graph Γ has the graph Δ *as an induced subgraph* if Γ has an induced subgraph that is isomorphic to Δ.

Graph monoids With each graph $\Gamma = (V, E)$, we associate the presentation $T_\Gamma = (X_\Gamma, R_\Gamma)$ over the alphabet $X_\Gamma = \{a_v, \bar{a}_v \mid v \in V\}$. The set $R_\Gamma \subseteq X_\Gamma^* \times X_\Gamma^*$ consists of the following pairs:

$$(a_v \bar{a}_v, \varepsilon) \qquad \text{for each } v \in V, \text{ and} \tag{2.2}$$

$$(xy, yx) \qquad \text{for each } x \in \{a_v, \bar{a}_v\},\ y \in \{a_w, \bar{a}_w\} \text{ with } \{v, w\} \in E. \tag{2.3}$$

This means in particular, we have $(a_v \bar{a}_v, \bar{a}_v a_v) \in R_\Gamma$ whenever $v \in V$ is looped. To simplify notation, the congruence $\equiv_{T_\Gamma}$ is also denoted by $\equiv_\Gamma$ and $[w]_{T_\Gamma}$ is also denoted $[w]_\Gamma$. With each graph Γ, we associate the monoid

$$\mathbb{M}\Gamma \;=\; X_\Gamma^* / \equiv_\Gamma .$$

Hence, $\mathbb{M}\Gamma$ is generated by $\{[a_v]_\Gamma, [\bar{a}_v]_\Gamma \mid v \in V\}$ and these elements always satisfy $[a_v]_\Gamma [\bar{a}_v]_\Gamma = 1$, where $1 = [\varepsilon]_\Gamma$ is the identity of $\mathbb{M}\Gamma$. Moreover, an edge $\{v, w\}$ means that the each element of $\{[a_v]_\Gamma, [\bar{a}_v]_\Gamma\}$ commutes with each element of $\{[a_w]_\Gamma, [\bar{a}_w]_\Gamma\}$. Monoids of the form $\mathbb{M}\Gamma$ are called *graph monoids*.

If every vertex of Γ is looped, we have $a_v \bar{a}_v \equiv_\Gamma \bar{a}_v a_v \equiv_\Gamma \varepsilon$ for $v \in V$ and thus $\mathbb{M}\Gamma$ is a group. Groups that arise as $\mathbb{M}\Gamma$ for such graphs are called *graph groups* [**Droms1987**, **LohreySteinberg2008**]. They are also known under the name *right-angled Artin groups* [**Charney2007**], *partially commutative groups* [**Diekert1990b**, **Wrathall1988**], or *semifree groups* [**Baudisch1981**]. For more information on these groups, see [**Charney2007**].

Examples Before we explain how concrete storage mechanisms can be realized by graph monoids, let us become familiar with how the graph structure impacts

the resulting monoid. If Γ consists of one unlooped vertex v, then R_Γ contains only the pair $(a_v \bar{a}_v, \varepsilon)$. By the definition of $\mathbb{B}$, we have $\mathbb{M}\Gamma \cong \mathbb{B}$ in this case. Moreover, if Γ consists of one looped vertex v, then it is easy to see that $\mathbb{M}\Gamma \cong \mathbb{Z}$.

Suppose $V = V_0 \uplus V_1$. Let $\Gamma_i = (V_i, E_i)$ be the subgraph of Γ induced by V_i for $i = 0, 1$. We write $X_i = X_{\Gamma_i}$. If for each $v_0 \in V_0$ and $v_1 \in V_1$, we have $\{v_0, v_1\} \in E$, then $[x_0 x_1]_\Gamma = [x_1 x_0]_\Gamma$ for $x_i \in X_i^*$ and hence the map

$$\begin{aligned} \mathbb{M}\Gamma_0 \times \mathbb{M}\Gamma_1 &\longrightarrow \mathbb{M}\Gamma, \\ ([x]_{\Gamma_0}, [y]_{\Gamma_1}) &\longmapsto [xy]_\Gamma \end{aligned}$$

is easily seen to be well-defined and an isomorphism. Furthermore, if there is no edge $\{v_0, v_1\} \in E$ with $v_0 \in V_0$ and $v_1 \in V_1$, then the map

$$\mathbb{M}\Gamma_0 * \mathbb{M}\Gamma_1 \to \mathbb{M}\Gamma$$

extending the maps $\mathbb{M}\Gamma_i \to \mathbb{M}\Gamma$ with $[x]_{\Gamma_i} \mapsto [x]_\Gamma$ for $i = 0, 1$ is again well-defined and an isomorphism.

Let $n = |V|$. The isomorphisms above imply that if Γ is a looped clique, then $\mathbb{M}\Gamma \cong \mathbb{Z}^n$. Furthermore, if Γ contains no edges, then $\mathbb{M}\Gamma \cong \mathbb{B}^{(n)}$. Finally, if Γ is a clique, then clearly $\mathbb{M}\Gamma \cong \mathbb{B}^n$.

Storage mechanisms as graph monoids The class of graph monoids accommodates several storage mechanisms. Here, we describe some examples. See Table 2.1 for an overview. Suppose Γ contains at least two vertices and no edges at all (not even loops). We have seen that then $\mathbb{M}\Gamma \cong \mathbb{B}^{(n)}$, where $n = |V|$. According to the explanation in Section 2.3, valence automata over $\mathbb{B}^{(n)}$ correspond to pushdown automata (with n stack symbols).

The fact that in this case, valence automata over $\mathbb{M}\Gamma$ are equivalent to pushdown automata can also be seen as follows: The identity language of $\mathbb{M}\Gamma$ with respect to the generating set $\{[x]_\Gamma \mid x \in X_\Gamma\}$ is the semi-Dyck language over the pairs $(a_v, \bar{a}_v)$ of parentheses. Hence, $\mathsf{VA}(\mathbb{M}\Gamma)$ consists of all rational transductions of a semi-Dyck language over at least two pairs of parentheses. By the theorem of Chomsky and Schützenberger (Theorem 2.1.3), $\mathsf{VA}(\mathbb{M}\Gamma)$ coincides with the context-free languages.

As a second example, suppose $\mathbb{M}\Gamma$ is a looped clique. We have seen above that then $\mathbb{M}\Gamma \cong \mathbb{Z}^n$, where $n = |V|$. It is obvious that in this case valence automata over such monoids $\mathbb{M}\Gamma$ are just a syntactic variant of blind counter automata.

For our third example, suppose Γ is a clique. As explained above, we have $\mathbb{M}\Gamma \cong \mathbb{B}^n$ for $n = |V|$. This means, $\mathbb{M}\Gamma$ is the direct product of n copies of the bicyclic monoid, each of which realizes a partially blind counter. Therefore, valence automata over $\mathbb{M}\Gamma$ are essentially partially blind multicounter automata.

Now suppose Γ is the graph C_4. Drawing C_4 as in Fig. 2.2c reveals that $\mathbb{M}\Gamma \cong \mathbb{B}^{(2)} \times \mathbb{B}^{(2)}$: The left pair of vertices and the right pair of vertices each forms the monoid $\mathbb{B}^{(2)}$. Moreover, there is an edge from each vertex on the left to each vertex on the right. Hence $\mathbb{M}\Gamma \cong \mathbb{B}^{(2)} \times \mathbb{B}^{(2)}$. Since taking the direct product of monoids corresponds to combining two storage mechanisms such that they can be used independently and simultaneously, valence automata over $\mathbb{M}\Gamma$ are automata with two pushdowns. Since a Turing tape can be realized using two pushdowns, it is clear that $\mathsf{VA}(\mathbb{M}\Gamma)$ is the class of recursively enumerable languages.

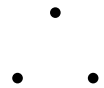
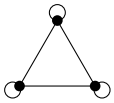
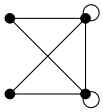
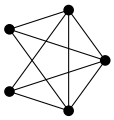
Graph Γ	Monoid $\mathbb{M}\Gamma$	Storage mechanism
	$\mathbb{B}^{(3)}$	Pushdown (with three symbols)
	$\mathbb{B}^3$	Three partially blind counters
	$\mathbb{Z}^3$	Three blind counters
	$\mathbb{B}^{(2)} \times \mathbb{Z}^2$	Pushdown (with two symbols) and two blind counters
	$\mathbb{B}^{(2)} \times \mathbb{B}^3$	Pushdown and three partially blind counters
	$\mathbb{B}^{(2)} \times \mathbb{B}^{(2)}$	Two pushdowns (with two symbols each)

Table 2.1: Examples of storage mechanisms

Of course, using direct products, we can realize every combination of storage mechanisms mentioned above. As a notable example, suppose $|V| = n > 2$ and Γ has no loops and is one edge short of being a clique. Then the two non-adjacent vertices $\{v, w\}$ together form the monoid $\mathbb{B}^{(2)}$ and $V \setminus \{v, w\}$ is a clique. Hence, we have $\mathbb{M}\Gamma \cong \mathbb{B}^{(2)} \times \mathbb{B}^{n-2}$. This means, a valence automaton over $\mathbb{M}\Gamma$ has access to a pushdown and $n-2$ partially blind counters. Such automata are, of course, equivalent to Petri nets with a pushdown, for which we will also use the term *pushdown Petri net*. It is a well-known open problem whether these models have a decidable reachability problem [**Reinhardt2008**].

If Γ consists of two non-adjacent looped vertices, then $\mathbb{M}\Gamma \cong \mathbb{Z} * \mathbb{Z}$. The expressive power of valence automata over $\mathbb{Z} * \mathbb{Z}$ is described by the following reformulation of the theorem of Chomsky and Schützenberger (Theorem 2.1.3). The reformulation has been proved by Corson [**Corson2005**] and Kambites [**Kambites2009**], the latter of whom observed the equivalence to the theorem of Chomsky and Schützenberger. The group $\mathbb{Z} * \mathbb{Z}$ is also known as the *free group of rank 2*.

Theorem 2.4.1 (Chomsky-Schützenberger/Corson/Kambites)**.** $\mathsf{VA}(\mathbb{Z} * \mathbb{Z})$ *equals the class of context-free languages.*

2.5 Semilinear intersections and powers of $\mathbb{Z}$

Since we will occasionally find ourselves in the situation that we understand the languages in $\mathsf{VA}(M)$ and want to describe the languages in $\mathsf{VA}(M \times \mathbb{Z}^n)$, this section introduces an operator on language classes that provides such a description. This means, in particular, we describe the effect of *adding blind counters* on the resulting language class.

Semilinear intersections Let $\mathcal{C}$ be a language class. Then $\mathsf{SLI}(\mathcal{C})$ denotes the class of languages of the form

$$h(L \cap \Psi^{-1}(S)),$$

where $L \subseteq X^*$ is in $\mathcal{C}$, the set $S \subseteq X^\oplus$ is semilinear, and $h\colon X^* \to Y^*$ is a morphism.

Proposition 2.5.1. *Let $\mathcal{C}$ be an effective full semi-AFL. Then $\mathsf{SLI}(\mathcal{C})$ is an effective Presburger closed full semi-AFL. In particular, $\mathsf{SLI}(\mathsf{SLI}(\mathcal{C})) = \mathsf{SLI}(\mathcal{C})$.*

Proof. Let $L \in \mathcal{C}$, $L \subseteq X^*$, $S \subseteq X^\oplus$ semilinear, and $h\colon X^* \to Y^*$ be a morphism. If $T \subseteq Y^* \times Z^*$ is a rational transduction, then $Th(L \cap \Psi^{-1}(S)) = U(L \cap \Psi^{-1}(S))$, where $U \subseteq X^* \times Z^*$ is the rational transduction

$$U = \{(u, v) \in X^* \times Z^* \mid (h(u), v) \in T\}.$$

We may assume that $X \cap Z = \emptyset$. Construct a regular language $R \subseteq (X \cup Z)^*$ with $U = \{(\pi_X(w), \pi_Z(w)) \mid w \in R\}$. With this, we have

$$U(L \cap \Psi^{-1}(S)) = \pi_Z\left((R \cap (L \sqcup Z^*)) \cap \Psi^{-1}(S + Z^\oplus)\right).$$

Since $\mathcal{C}$ is an effective full semi-AFL, and thus $R \cap (L \sqcup Z^*)$ is effectively in $\mathcal{C}$, the right-hand side is effectively contained in $\mathsf{SLI}(\mathcal{C})$. This proves that $\mathsf{SLI}(\mathcal{C})$ is an effective full trio.

Let us prove effective closure under union. Now suppose $L_i \subseteq X_i^*$, $S_i \subseteq X_i^\oplus$, and $h\colon X_i^* \to Y^*$ for $i = 1, 2$. If $\bar{X}_2$ is a disjoint copy of X_2 with bijection $\varphi\colon X_2 \to \bar{X}_2$, then

$$h_1(L_1 \cap \Psi^{-1}(S_1)) \cup h_2(L_2 \cap \Psi^{-1}(S_2)) = h((L_1 \cup \varphi(L_2)) \cap \Psi^{-1}(S_1 \cup \varphi(S_2))),$$

where $h\colon X_1 \cup \bar{X}_2 \to Y$ is the map for which $h(x) = h_1(x)$ for $x \in X_1$ and $h(x) = h_2(\varphi(x))$ for $x \in \bar{X}_2$. This proves that $\mathsf{SLI}(\mathcal{C})$ is effectively closed under union.

It remains to be shown that $\mathsf{SLI}(\mathcal{C})$ is Presburger closed. Suppose $L \in \mathcal{C}$, $L \subseteq X^*$, $S \subseteq X^\oplus$ is semilinear, $h\colon X^* \to Y^*$ is a morphism, and $T \subseteq Y^\oplus$ is another semilinear set. Let $\varphi\colon X^\oplus \to Y^\oplus$ be the morphism with $\varphi(\Psi(w)) = \Psi(h(w))$ for every $w \in X^*$. Moreover, consider the set

$$T' = \{\mu \in X^\oplus \mid \varphi(w) \in T\} = \{\Psi(w) \mid w \in X^*,\ \Psi(h(w)) \in T\}.$$

It is clearly Presburger definable (because T is) and hence effectively semilinear. Furthermore, we have

$$h(L \cap \Psi^{-1}(S)) \cap \Psi^{-1}(T) = h(L \cap \Psi^{-1}(S \cap T')).$$

This proves that $\mathsf{SLI}(\mathcal{C})$ is effectively Presburger closed. □

We will often use the following fact to show that certain language classes are semilinear.

Proposition 2.5.2. *If $\mathcal{C}$ is semilinear, then so is $\mathsf{SLI}(\mathcal{C})$. Moreover, if $\mathcal{C}$ is effectively semilinear, then so is $\mathsf{SLI}(\mathcal{C})$.*

Proof. Since morphisms effectively preserve semilinearity, it suffices to show that $\Psi(L \cap \Psi^{-1}(S))$ is (effectively) semilinear for each $L \in \mathcal{C}$, $L \subseteq X^*$, and semilinear $S \subseteq X^\oplus$. This, however, is easy to see since $\Psi(L \cap \Psi^{-1}(S)) = \Psi(L) \cap S$ and the semilinear subsets of $X^\oplus$ are closed under intersection (they coincide with the Presburger definable sets). Furthermore, if a semilinear representation of $\Psi(L)$ can be computed, this is also the case for $\Psi(L) \cap S$. □

We are now ready for our description of the languages in $\mathsf{VA}(M \times \mathbb{Z}^n)$ in terms of $\mathsf{VA}(M)$ and the operator $\mathsf{SLI}(\cdot)$.

Proposition 2.5.3. *Let M be a monoid. Then $\mathsf{SLI}(\mathsf{VA}(M)) = \bigcup_{n \geqslant 0} \mathsf{VA}(M \times \mathbb{Z}^n)$. Moreover, this equality is effective.*

Proof. We start with the inclusion "$\subseteq$". Since the right-hand side is closed under morphisms and union, it suffices to show that for each $L \in \mathsf{VA}(M)$, $L \subseteq X^*$, and semilinear $S \subseteq X^\oplus$, we have $L \cap \Psi^{-1}(S) \in \mathsf{VA}(M \times \mathbb{Z}^n)$ for some $n \geqslant 0$. Let $n = |X|$ and pick a linear order on X. This induces an embedding $X^\oplus \to \mathbb{Z}^n$, by way of which we consider $X^\oplus$ as a subset of $\mathbb{Z}^n$.

Suppose $L = L(A)$ for a valence automaton A over M. The new valence automaton A' over $M \times \mathbb{Z}^n$ simulates A and, if w is the input read by A, adds $\Psi(w)$ to the $\mathbb{Z}^n$ component of the storage monoid. When A reaches a final state, A' nondeterministically changes to a new state q_1, in which it nondeterministically subtracts an element of S from the $\mathbb{Z}^n$ component. Afterwards, A' switches to another new state q_2, which is the only accepting state in A'. Clearly, A' accepts

a word w if and only if $w \in L(A)$ and $\Psi(w) \in S$, hence $L(A') = L(A) \cap \Psi^{-1}(S)$. This proves "$\subseteq$".

Suppose $L = L(A)$ for some valence automaton $A = (Q, X, M \times \mathbb{Z}^n, E, q_0, F)$. We construct a valence automaton A' over M as follows. The input alphabet X' of A' is the set of pairs $(w, \mu) \in X^* \times \mathbb{Z}^n$ for which there is an edge $(p, w, (m, \mu), q)$ in E for some $p, q \in Q$, $m \in M$. A' has edges

$$E' = \{(p, (w, \mu), m, q) \mid (p, w, (m, \mu), q) \in E\}.$$

In other words, whenever A reads w and adds $(m, \mu) \in M \times \mathbb{Z}^n$ to its storage monoid, A' adds m and reads (w, μ) from the input. Let $\psi\colon X'^{\oplus} \to \mathbb{Z}^n$ be the morphism that projects the symbols in X' to the right component and let $h\colon X'^* \to X^*$ be the morphism that projects the symbols in X' to the left component. Note that the set $S = \psi^{-1}(0) \subseteq X'^{\oplus}$ is Presburger definable and hence effectively semilinear. We clearly have $L(A) = h(L(A') \cap \Psi^{-1}(S)) \in \mathsf{SLI}(\mathsf{VA}(M))$. This proves "$\supseteq$". Clearly, all constructions in the proof can be carried out effectively. □

2.6 Algebraic extensions

This section is devoted to a language theoretic concept, algebraic extensions. They will be used here to describe how taking the free product (and a more general product construction) affects the class of languages accepted by valence automata. In particular, we will describe the effect of *building stacks* on the resulting language class.

Algebraic extensions were invented by van Leeuwen [**vanLeeuwen1974**] in an effort to generalize Parikh's theorem (see Theorem 2.6.7).

Algebraic extensions Let $\mathcal{C}$ be a class of languages. A *$\mathcal{C}$-grammar* is a quadruple $G = (N, T, P, S)$ where N and T are disjoint alphabets and $S \in N$. The symbols in N and T are called the *nonterminals* and the *terminals*, respectively. P is a finite set of pairs (A, M) with $A \in N$ and $M \subseteq (N \cup T)^*$, $M \in \mathcal{C}$. A pair $(A, M) \in P$ is called a *production of* G and also denoted by $A \to M$. The set M is the *right-hand side* of the production $A \to M$.

We write $x \Rightarrow_G y$ if $x = uAv$ and $y = uwv$ for some $u, v, w \in (N \cup T)^*$ and $(A, M) \in P$ with $w \in M$. A word w with $S \Rightarrow_G^* w$ is called a *sentential form* of G and we write $\mathsf{SF}(G)$ for the set of sentential forms of G. The *language generated by* G is $L(G) = \mathsf{SF}(G) \cap T^*$. Languages generated by $\mathcal{C}$-grammars are called *algebraic over* $\mathcal{C}$. The class of all languages that are algebraic over $\mathcal{C}$ is called the *algebraic extension* of $\mathcal{C}$ and denoted $\mathsf{Alg}(\mathcal{C})$. We say a language class $\mathcal{C}$ is *algebraically closed* if $\mathsf{Alg}(\mathcal{C}) = \mathcal{C}$. The grammars G and H are *equivalent* if $L(G) = L(H)$. If $\mathcal{C}$ is the class of finite languages, $\mathcal{C}$-grammars are also called *context-free grammars* and are known to generate precisely the context-free languages.

A nonterminal $A \in N$ is called *reachable* if there are $u, v \in (N \cup T)^*$ with $S \Rightarrow_G^* uAv$. It is said to be *productive* if there is a word $w \in T^*$ with $A \Rightarrow_G^* w$. The $\mathcal{C}$-grammar G is called *reduced* if each of its nonterminals is reachable and productive. Of course, every $\mathcal{C}$-grammar has a reduced equivalent. If $\mathcal{C}$ is an effective full semi-trio and emptiness is decidable for languages in $\mathcal{C}$, then a reduced equivalent can be determined effectively: First, we compute the set of

productive nonterminals. We initialize $N_0 = \emptyset$ and then successively compute

$$N_{i+1} = \{A \in N \mid L \cap (N_i \cup T)^* \neq \emptyset \text{ for some } A \to L \text{ in } P\}.$$

Then at some point, $N_{i+1} = N_i$ and N_i contains precisely the productive nonterminals. Using a similar method, one can compute the set of reachable nonterminals. Hence, one can compute the set $N' \subseteq N$ of nonterminals that are reachable and productive. The new grammar is then obtained by replacing each production $A \to L$ with $A \to (L \cap (N' \cup T)^*)$ and removing all productions $A \to L$ where $A \notin N'$.

Our first observation concerning algebraic extensions is that they have useful closure properties, provided that $\mathcal{C}$ satisfies some mild closure properties.

Proposition 2.6.1. *Let $\mathcal{C}$ be an effective full semi-trio. Then $\mathsf{Alg}(\mathcal{C})$ is an effective full semi-AFL.*

Proof. Since $\mathsf{Alg}(\mathcal{C})$ is clearly effectively closed under union, we only prove effective closure under rational transductions.

Let $G = (N, T, P, S)$ be a $\mathcal{C}$-grammar and let $U \subseteq T^* \times X^*$ be a rational transduction. Since we can easily construct a $\mathcal{C}$-grammar for $aL(G)$ (just add a production $S' \to \{aS\}$) and the rational transduction $(a, \varepsilon)U = \{(au, v) \mid (u, v) \in U\}$, we may assume that $L(G) \subseteq T^+$.

Let U be given by the automaton $A = (Q, T, X, E, q_0, F)$. We may assume that

$$E \subseteq Q \times ((T \times \{\varepsilon\}) \cup (\{\varepsilon\} \times X)) \times Q$$

and $F = \{f\}$. We regard $Z = Q \times T \times Q$ and $N' = Q \times N \times Q$ as alphabets. For each $p, q \in Q$, let $U_{p,q} \subseteq (N \cup T)^* \times (N' \cup Z)^*$ be the transduction such that for $w = w_1 \cdots w_n$, $w_1, \ldots, w_n \in N \cup T$, $n \geqslant 1$, the set $U_{p,q}(w)$ consists of all words

$$(p, w_1, q_1)(q_1, w_2, q_2) \cdots (q_{n-1}, w_n, q)$$

with $q_1, \ldots, q_{n-1} \in Q$. Moreover, let $U_{p,q}(\varepsilon) = \{\varepsilon\}$ if $p = q$ and $U_{p,q}(\varepsilon) = \emptyset$ if $p \neq q$. Observe that $U_{p,q}$ is a locally finite rational transduction. The new grammar $G' = (N', Z, P', (q_0, S, f))$ has productions $(p, B, q) \to U_{p,q}(L)$ for each $p, q \in Q$ and $B \to L \in P$. Let $\sigma\colon Z^* \to \mathcal{P}(X^*)$ be the regular substitution defined by

$$\sigma((p, x, q)) = \{w \in X^* \mid (p, (\varepsilon, \varepsilon)) \to_A^* (q, (x, w))\}.$$

We claim that $U(L(G)) = \sigma(L(G'))$. First, it can be shown by induction on the number of derivation steps that $\mathsf{SF}(G') = U_{q_0,f}(\mathsf{SF}(G))$. In particular, we have the equation $L(G') = U_{q_0,f}(L(G))$. Since for every language $K \subseteq T^+$, we have $\sigma(U_{q_0,f}(K)) = UK$, we may conclude $\sigma(L(G')) = U(L(G))$.

$\mathsf{Alg}(\mathcal{C})$ is clearly effectively closed under $\mathsf{Alg}(\mathcal{C})$-substitutions. Since $\mathcal{C}$ contains the finite languages, this means $\mathsf{Alg}(\mathcal{C})$ is closed under Reg-substitutions. Hence, we can construct a $\mathcal{C}$-grammar for $U(L(G)) = \sigma(L(G'))$. □

2.6.1 Free products with amalgamation

As mentioned above, we wish to describe the languages in $\mathsf{VA}(M_0 * M_1)$ in terms of algebraic extensions and of the languages in $\mathsf{VA}(M_0)$ and $\mathsf{VA}(M_1)$. In fact, this description is also available for a more general product, in which the two factors

are "glued together" along a common submonoid. This product is called free product with amalgamation. It will be used in Chapter 6, which characterizes those graph products that admit only context-free languages.

Let M_0, M_1 be monoids with $M_0 \cap M_1 = \emptyset$ and let $\theta_i \colon N \to M_i$ be an injective morphism for each $i \in \{0,1\}$. Moreover, let $\approx$ be the smallest congruence in $M_0 * M_1$ such that $\theta_0(a) \approx \theta_1(a)$ for every $a \in N$. Then the monoid

$$M_0 *_N M_1 = (M_0 * M_1)/\approx$$

is called a *free product with amalgamation*. Since the notation $M_0 *_N M_1$ does not make the maps θ_0, θ_1 explicit, they will always be clear from the context. Clearly, if N is the trivial monoid, then $M_0 *_N M_1 \cong M_0 * M_1$.

If $[m]$ denotes the equivalence class of $m \in M_0 *_N M_1$ with respect to $\approx$, then the morphisms $\psi_i \colon M_i \to M_0 *_N M_1$ for $i \in \{0,1\}$ with $\psi_i(m) = [m]$ for $m \in M_i$ are called the *canonical morphisms*. Here, we will only be concerned with the situation where N is a group, in which case the following holds. For a proof, we refer the reader to **[LohreySenizergues2008]**.

Lemma 2.6.2. *Let $\theta_i \colon N \to M_i$ for $i \in \{0,1\}$ be injective morphisms. If N is a group, then the following holds.*

1. *The canonical morphisms $M_i \to M_0 *_N M_1$ are injective.*
2. *Suppose $t_1, \ldots, t_n \in M_0 \cup M_1$ such that $t_j \in M_i$ if and only if $t_{j+1} \in M_{1-i}$ for $1 \leqslant j < n$ and $i \in \{0,1\}$. Then $t_1 \cdots t_n \approx 1$ implies that for some index $1 \leqslant j \leqslant n$, we have $t_j \in \theta_0(N) \cup \theta_1(N)$.*

We are now ready to describe the languages in $\mathsf{VA}(M_0 *_N M_1)$ in terms of languages in $\mathsf{VA}(M_0)$ and $\mathsf{VA}(M_1)$. The following result appeared in **[BuckheisterZetzsche2013a]** and in the special case of the free product in **[Zetzsche2013a]**.

Theorem 2.6.3. *Let F be a finite group and $\theta_i \colon F \to M_i$ be an injective morphism[6] for $i = 0, 1$. Then $\mathsf{VA}(M_0 *_F M_1) \subseteq \mathsf{Alg}(\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1))$.*

We will see later (Theorem 2.6.7) that algebraic extensions preserve semilinearity. Therefore, Theorem 2.6.3 implies that the properties of $\mathsf{VA}(M)$ being semilinear or context-free are each preserved under taking free products with amalgamation over a finite identified subgroup. In the case where the factors are residually finite groups, the preservation of semilinearity was already shown by **LohreySteinberg2008** [**LohreySteinberg2008**].

Proof of Theorem 2.6.3. Since the algebraic extension of a full trio is again a full trio (Proposition 2.6.1), it suffices to show that with respect to some generating set $S \subseteq M_0 *_F M_1$, the identity language of $M_0 *_F M_1$ is algebraic over $\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1)$.

For $i \in \{0,1\}$, let $S_i \subseteq M_i$ be a finite generating set for M_i such that $\theta_i(F) \subseteq S_i$. Furthermore, let X_i be an alphabet in bijection with S_i and let $\varphi_i \colon X_i^* \to M_i$ be the morphism extending this bijection. Moreover, let $Y_i \subseteq X_i$ be the subset with $\varphi_i(Y_i) = \theta_i(F)$. Let $\psi_i \colon M_i \to M_0 *_F M_1$ be the canonical morphism. Since F is a group, ψ_0 and ψ_1 are injective by Lemma 2.6.2. Let $X = X_0 \cup X_1$ and

[6] Recall that here, morphisms are always monoid morphisms (as opposed to semigroup morphisms), which means that $\theta_i(1)$ has to be the identity of M_i.

let $\varphi\colon X^* \to M_0 *_F M_1$ be the morphism extending $\psi_0\varphi_0$ and $\psi_1\varphi_1$. Then the identity language of $M_0 *_F M_1$ is $\varphi^{-1}(1)$ and we shall prove the theorem by showing that $\varphi^{-1}(1)$ is algebraic over the language class $\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1)$.

For each $i \in \{0,1\}$ and $f \in F$, we define $L_{i,f} = \varphi_i^{-1}(\theta_i(f))$ and write $y_{i,f}$ for the symbol in Y_i with $\varphi_i(y_{i,f}) = \theta_i(f^{-1})$. Since $\theta_i(1) = 1$, we have $L_{i,1} \in \mathsf{VA}(M_i)$. Furthermore, since F is a group, the equation $\varphi_i(w) = \theta_i(f)$ is equivalent to $\theta_i(f^{-1})\varphi_i(w) = \theta_i(1)$. This means we have

$$L_{i,f} = \{w \in X_i^* \mid y_{i,f}w \in L_{i,1}\}$$

and can thus obtain $L_{i,f}$ from $L_{i,1}$ using a rational transduction. Hence, we have $L_{i,f} \in \mathsf{VA}(M_i)$.

Let $\mathcal{C} = \mathsf{VA}(M_0) \cup \mathsf{VA}(M_1)$. Since for each $\mathcal{C}$-grammar G, it is clearly possible to construct a $\mathcal{C}$-grammar G' such that $L(G')$ consists of all sentential forms of G, it suffices to construct a $\mathcal{C}$-grammar $G = (N, T, P, S)$ with $N \cup T = X$ and $S \Rightarrow_G^* w$ if and only if $\varphi(w) = 1$ for $w \in X^*$. We construct $G = (N, T, P, S)$ as follows. Let $N = Y_0 \cup Y_1$ and $T = (X_0 \cup X_1) \setminus (Y_0 \cup Y_1)$. As productions, we have $y \to L_{1-i,f}$ for each $y \in Y_i$, where $\theta_i(f) = \varphi_i(y)$. Since $1 \in \theta_i(F)$, there is an $e_i \in Y_i$ with $\varphi_i(e_i) = 1$. As the start symbol, we choose $S = e_0$. We claim that for $w \in X^*$, we have $S \Rightarrow_G^* w$ if and only if $\varphi(w) = 1$.

The "only if" follows from the fact that $\varphi(u) = \varphi(v)$ whenever $u \Rightarrow_G v$ for $u, v \in X^*$. Thus, let $w \in X^*$ with $\varphi(w) = 1$. We write $w = w_1 \cdots w_n$ such that $w_j \in X_0^* \cup X_1^*$ for all $1 \leqslant j \leqslant n$ such that $w_j \in X_i^*$ if and only if $w_{j+1} \in X_{1-i}^*$ for $i \in \{0,1\}$ and $1 \leqslant j < n$. We show by induction on n that $S \Rightarrow_G^* w$. For $n \leqslant 1$, we have $w \in X_i^*$ for some $i \in \{0,1\}$. Since $1 = \varphi(w) = \psi_i(\varphi_i(w))$ and ψ_i is injective, we have $\varphi_i(w) = 1 = \theta_i(1)$ and hence $w \in L_{i,1}$. This means $S = e_0 \Rightarrow_G w$ or $S = e_0 \Rightarrow_G e_1 \Rightarrow_G w$, depending on whether $i = 1$ or $i = 0$.

Now let $n \geqslant 2$. Since $\varphi(w_1 \cdots w_n) = 1$, Lemma 2.6.2 provides a $j \in \{1, \ldots, n\}$ and an $i \in \{0,1\}$ with $w_j \in X_i^*$ and $\varphi_i(w_j) \in \theta_i(F)$. Thus, with $\theta_i(f) = \varphi(w_j)$ we have $w_j \in L_{i,f}$. Hence, if we choose $y \in Y_{1-i}$ with $\varphi_{1-i}(y) = \theta_i(f)$, then $w' = w_1 \cdots w_{j-1} y w_{j+1} \cdots w_n \Rightarrow_G w$. For w' the induction hypothesis holds, meaning $S \Rightarrow_G^* w'$ and thus $S \Rightarrow_G^* w$. □

Theorem 2.6.3 tells us that the languages in $\mathsf{VA}(M_0 *_F M_1)$ are confined to the algebraic extension of $\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1)$. The rest of this section complements Theorem 2.6.3 by describing monoids N such that the algebraic extension of $\mathsf{VA}(M)$ is confined to $\mathsf{VA}(N)$. We need two auxiliary lemmas, for which the following notation will be convenient. We write $M \hookrightarrow N$ for monoids M, N if there is a morphism $\varphi\colon M \to N$ such that $\varphi^{-1}(1) = \{1\}$. Clearly, if $M \hookrightarrow N$, then $\mathsf{VA}(M) \subseteq \mathsf{VA}(N)$: Replacing in a valence automaton over M all elements $m \in M$ with $\varphi(m)$ yields a valence automaton over N that accepts the same language.

Lemma 2.6.4. *If $M \hookrightarrow M'$ and $N \hookrightarrow N'$, then we have $M * N \hookrightarrow M' * N'$.*

Proof. Suppose $\varphi\colon M \to M'$ and $\psi\colon N \to N'$ are morphisms with $\varphi^{-1}(1) = \{1\}$ and $\psi^{-1}(1) = \{1\}$. Then defining $\kappa\colon M * N \to M' * N'$ as the morphism with $\kappa|_M = \varphi$ and $\kappa|_N = \psi$ clearly yields $\kappa^{-1}(1) = 1$. □

Lemma 2.6.5. *Let M be a monoid with $R_1(M) \neq \{1\}$. Then $\mathbb{B}^{(n)} * M \hookrightarrow \mathbb{B} * M$ for every $n \geqslant 1$. In particular, $\mathsf{VA}(\mathbb{B} * M) = \mathsf{VA}(\mathbb{B}^{(n)} * M)$ for every $n \geqslant 1$.*

Proof. Observe that if $\mathbb{B}^{(n)} * M \hookrightarrow \mathbb{B} * M$ and $\mathbb{B} * \mathbb{B} * M \hookrightarrow \mathbb{B} * M$, then

$$\mathbb{B}^{(n+1)} * M \cong \mathbb{B} * (\mathbb{B}^{(n)} * M) \hookrightarrow \mathbb{B} * (\mathbb{B} * M) \hookrightarrow \mathbb{B} * M.$$

Therefore, it suffices to prove $\mathbb{B} * \mathbb{B} * M \hookrightarrow \mathbb{B} * M$.

Let $\mathbb{B}_s = \langle s, \bar{s} \mid s\bar{s} = 1 \rangle$ for $s \in \{p, q, r\}$. We show $\mathbb{B}_p * \mathbb{B}_q * M \hookrightarrow \mathbb{B}_r * M$. Suppose M is presented by (X, R). We regard the monoids $\mathbb{B}_p * \mathbb{B}_q * M$ and $\mathbb{B}_r * M$ as embedded into $\mathbb{B}_p * \mathbb{B}_q * \mathbb{B}_r * M$, which by definition of the free product, has a presentation (Y, S), where $Y = \{p, \bar{p}, q, \bar{q}, r, \bar{r}\} \cup X$ and S consists of R and the equations $s\bar{s} = 1$ for $s \in \{p, q, r\}$. For $w \in Y^*$, we write $[w]$ for the class of w in the congruence generated by S. Since $R_1(M) \neq \{1\}$, we find $u, v \in X^*$ with $[uv] = 1$ and $[u] \neq 1$. and let $\varphi \colon (\{p, \bar{p}, q, \bar{q}\} \cup X)^* \to (\{r, \bar{r}\} \cup X)^*$ be the morphism with $\varphi(x) = x$ for $x \in X$ and

$$\begin{aligned} p &\mapsto rr, & \bar{p} &\mapsto \bar{r}\bar{r}, \\ q &\mapsto rur, & \bar{q} &\mapsto \bar{r}v\bar{r}. \end{aligned}$$

We show by induction on $|w|$ that $[\varphi(w)] = 1$ implies $[w] = 1$. Since this is trivial for $w = \varepsilon$, we assume $|w| \geqslant 1$. Now suppose $[\varphi(w)] = [\varepsilon]$ for some $w \in (\{p, \bar{p}, q, \bar{q}\} \cup X)^*$. If $w \in X^*$, then $[\varphi(w)] = [w]$ and hence $[w] = 1$. Otherwise, we have $\varphi(w) = xry\bar{r}z$ for some $y \in X^*$ with $[y] = 1$ and $[xz] = 1$. This means $w = fsys'g$ for $s, s' \in \{p, q\}$ with $\varphi(fs) = xr$ and $\varphi(\overline{s'}g) = \bar{r}z$. If $s \neq s'$, then $s = p$ and $s' = q$; or $s = q$ and $s' = p$. In the former case

$$[\varphi(w)] = [\varphi(f)\ rr\ y\ \bar{r}v\bar{r}\ \varphi(g)] = [\varphi(f)rv\bar{r}\varphi(g)] \neq 1$$

since $[v] \neq 1$ and in the latter

$$[\varphi(w)] = [\varphi(f)\ rur\ y\ \bar{r}\bar{r}\ \varphi(g)] = [\varphi(f)ru\bar{r}\varphi(g)] \neq 1$$

since $[u] \neq 1$. Hence $s = s'$. This means $1 = [w] = [fsy\bar{s}g] = [fg]$ and also $1 = [\varphi(w)] = [\varphi(fg)]$ and since $|fg| < |w|$, induction yields $[w] = [fg] = 1$.

Hence, we have shown that $[\varphi(w)] = 1$ implies $[w] = 1$. Since, on the other hand, $[u] = [v]$ implies $[\varphi(u)] = [\varphi(v)]$ for all $u, v \in (\{p, \bar{p}, q, \bar{q}\} \cup X)^*$, we can lift φ to a morphism witnessing $\mathbb{B}_p * \mathbb{B}_q * M \hookrightarrow \mathbb{B}_r * M$. □

The following result is the announced counterpart of Theorem 2.6.3. Observe that since $\mathsf{VA}(\mathbb{B} * \mathbb{B})$ is the class of languages accepted by pushdown automata and $\mathsf{Alg}(\mathsf{Reg}) = \mathsf{Alg}(\mathsf{VA}(\mathbf{1}))$ is clearly the class of languages generated by context-free grammars, its first statement generalizes the equivalence between pushdown automata and context-free grammars. Moreover, the second statement describes the language class that is obtained when we go from one storage mechanism to the other by *building stacks*.

Theorem 2.6.6. *For every monoid* M, $\mathsf{VA}(\mathbb{B} * \mathbb{B} * M) = \mathsf{Alg}(\mathsf{VA}(M))$. *Moreover, if* $R_1(M) \neq \{1\}$, *then* $\mathsf{VA}(\mathbb{B} * M) = \mathsf{Alg}(\mathsf{VA}(M))$.

Proof. It suffices to prove the first statement: If $R_1(M) \neq \{1\}$, then Lemma 2.6.5 implies $\mathsf{VA}(\mathbb{B} * M) = \mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$. Since $\mathsf{VA}(\mathbb{B}) \subseteq \mathsf{CF}$, Theorem 2.6.3 yields

$$\mathsf{VA}(\mathbb{B} * N) \subseteq \mathsf{Alg}(\mathsf{VA}(\mathbb{B}) \cup \mathsf{VA}(N)) \subseteq \mathsf{Alg}(\mathsf{VA}(N))$$

for every monoid N. Therefore,

$$\mathsf{VA}(\mathbb{B} * \mathbb{B} * M) \subseteq \mathsf{Alg}(\mathsf{VA}(\mathbb{B} * M)) \subseteq \mathsf{Alg}(\mathsf{Alg}(\mathsf{VA}(M))) = \mathsf{Alg}(\mathsf{VA}(M)).$$

It remains to be shown that $\mathsf{Alg}(\mathsf{VA}(M)) \subseteq \mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$.

Suppose $G = (N, T, P, S)$ is a reduced $\mathsf{VA}(M)$-grammar and let $X = N \cup T$. Since $\mathsf{VA}(M)$ is closed under union, we may assume that for each $B \in N$, there is precisely one production $B \to L_B$ in P. For each nonterminal $B \in N$, there is a valence automaton $A_B = (Q_B, X, M, E_B, q_0^B, F_B)$ over M with $L(A_B) = L_B$. We may clearly assume that $Q_B \cap Q_C = \emptyset$ for $B \neq C$ and that for each $(p, w, m, q) \in E_B$, we have $|w| \leqslant 1$.

In order to simplify the correctness proof, we modify G. Let $\lfloor$ and $\rfloor$ be new symbols and let G' be the grammar $G' = (N, T \cup \{\lfloor, \rfloor\}, P', S)$, where P' consists of the productions $B \to \lfloor L \rfloor$ for $B \to L \in P$. Moreover, let

$$K = \{v \in (N \cup T \cup \{\lfloor, \rfloor\})^* \mid u \Rightarrow_G^* v,\ u \in L_S\}.$$

Then $L(G) = \pi_T(K \cap (T \cup \{\lfloor, \rfloor\})^*)$ and it suffices to show $K \in \mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$.

Let $Q = \bigcup_{B \in N} Q_B$. For each $q \in Q$, let $\mathbb{B}_q = \langle q, \bar{q} \mid q\bar{q} = 1 \rangle$ be an isomorphic copy of $\mathbb{B}$. Let $M' = \mathbb{B}_{q_1} * \cdots * \mathbb{B}_{q_n} * M$, where $Q = \{q_1, \ldots, q_n\}$. We shall prove $K \in \mathsf{VA}(M')$, which implies $K \in \mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$ by Lemma 2.6.5 since $R_1(\mathbb{B} * M) \neq \{1\}$.

Let $E = \bigcup_{B \in N} E_B$, $F = \bigcup_{B \in N} F_B$. The new set E' consists of the following transitions:

$$(p, x, m, q) \qquad \text{for } (p, x, m, q) \in E, \tag{2.4}$$

$$(p, \lfloor, mq, q_0^B) \qquad \text{for } (p, B, m, q) \in E,\ B \in N, \tag{2.5}$$

$$(p, \rfloor, \bar{q}, q) \qquad \text{for } p \in F,\ q \in Q. \tag{2.6}$$

We claim that with $A' = (Q, N \cup T \cup \{\lfloor, \rfloor\}, M', E', q_0^S, F)$, we have $L(A') = K$.

Let $v \in K$, where $u \Rightarrow_G^n v$ for some $u \in L_S$. We show $v \in L(A')$ by induction on n. For $n = 0$, we have $v \in L_S$ and can use transitions of type (2.4) inherited from A_S to accept v. If $n \geqslant 1$, let $u \Rightarrow_{G'}^{n-1} v' \Rightarrow_{G'} v$. Then $v' \in L(A')$ and $v' = xBy$, $v = x\lfloor w \rfloor y$ for some $B \in N$, $w \in L_B$. The run for v' uses a transition $(p, B, m, q) \in E$. Instead of using this transition, we can use $(p, \lfloor, mq, q_0^B)$, then execute the (2.4)-type transitions for $w \in L_B$, and finally use $(f, \rfloor, \bar{q}, q)$, where f is the final state in the run for w. This has the effect of reading $\lfloor w \rfloor$ from the input and multiplying $mq1\bar{q} = m$ to the storage monoid. Hence, the new run is valid and accepts v. Hence, $v \in L(A')$. This proves $K \subseteq L(A')$.

In order to show $L(A') \subseteq K$, consider the morphisms $\varphi\colon (T \cup \{\lfloor, \rfloor\})^* \to \mathbb{B}$, $\psi\colon M' \to \mathbb{B}$ with $\varphi(x) = 1$ for $x \in T$, $\varphi(\lfloor) = a$, $\varphi(\rfloor) = \bar{a}$, $\psi(q) = a$ for $q \in Q$, $\psi(\bar{q}) = \bar{a}$, and $\psi(m) = 1$ for $m \in M$. The transitions of A' are constructed such that $(p, \varepsilon, 1) \to_{A'}^* (q, w, m)$ implies $\varphi(w) = \psi(m)$. In particular, if $v \in L(A')$, then $\pi_{\{\lfloor, \rfloor\}}(v)$ is a semi-Dyck word with respect to $\lfloor$ and $\rfloor$.

Let $v \in L(A')$ and let $n = |w|_{\lfloor}$. We show $v \in K$ by induction on n. If $n = 0$, then the run for v only used transitions of type (2.4) and hence $v \in L_S$. If $n \geqslant 1$, since $\pi_{\{\lfloor, \rfloor\}}(v)$ is a semi-Dyck word, we can write $v = x\lfloor w \rfloor y$ for some $w \in (N \cup T)^*$. Since $\lfloor$ and $\rfloor$ can only be produced by transitions of the form (2.5)

and (2.6), respectively, the run for ν has to be of the form

$$\begin{aligned}(q_0^S, \varepsilon, 1) &\to_{A'}^* (p, x, r)\\ &\to_{A'} (q_0^B, x\lfloor, rmq)\\ &\to_{A'}^* (f, x\lfloor w, rmqs)\\ &\to_{A'} (q', x\lfloor w\rfloor, rmqs\overline{q'})\\ &\to_{A'}^* (f', x\lfloor w\rfloor y, rmqs\overline{q'}t)\end{aligned}$$

for some $p, q, q' \in Q$, $B \in N$, $(p, B, m, q) \in E$, $f, f' \in F$, $r, t \in M'$, and $s \in M$ and with $rmqs\overline{q'}t = 1$. This last condition implies $s = 1$ and $q = q'$, which in turn entails $rmt = 1$. This also means $(p, B, m, q') = (p, B, m, q) \in E$ and $(q_0^B, \varepsilon, 1) \to_{A'}^* (f, w, s) = (f, w, 1)$ and hence $w \in L_B$. Using the transition $(p, B, m, q') \in E$, we have

$$\begin{aligned}(q_0^S, \varepsilon, 1) &\to_{A'}^* (p, x, r)\\ &\to_{A'} (q', xB, rm)\\ &\to_{A'}^* (f', xBy, rmt).\end{aligned}$$

Hence $xBy \in L(A')$ and $|xBy|_\lfloor < |\nu|_\lfloor$. Thus, induction yields $xBy \in K$ and since $xBy \Rightarrow_{G'} x\lfloor w\rfloor y$, we have $\nu = x\lfloor w\rfloor y \in K$. This proves $L(A') = K$. □

2.6.2 Parikh images

When establishing semilinearity of certain classes $VA(M)$, we will often rely on the following result of van Leeuwen. Essentially the same fact was shown by Greibach in her work **[Greibach1972]**, which formulates it using nested iterated substitutions instead of algebraic extensions.

Theorem 2.6.7 (van Leeuwen **[vanLeeuwen1974]**). *If $\mathcal{C}$ is semilinear, then so is $\mathsf{Alg}(\mathcal{C})$.*

On the one hand, this can be derived from Parikh's Theorem: In a $\mathcal{C}$-grammar, replace each right-hand side by a Parikh equivalent regular language; this is possible since $\mathcal{C}$ is semilinear. The language of the new grammar is Parikh equivalent and context-free. Hence, it is semilinear by Parikh's Theorem. On the other hand, van Leeuwen deduces it from a (slight variant) of the following relative version.

Theorem 2.6.8 (**vanLeeuwen1974 [vanLeeuwen1974]**). *For each substitution closed full semi-AFL $\mathcal{C}$, we have $\Psi(\mathsf{Alg}(\mathcal{C})) = \Psi(\mathcal{C})$. Moreover, if $\mathcal{C}$ exhibits these closure properties effectively, then this equality is effective*[7] *as well.*

The generality of Theorem 2.6.8 allowed van Leeuwen to provide a very elegant inductive proof. Since the same principle will shape the construction of Parikh annotations in Section 9.2, this proof is instructive and therefore included here. Its key idea is a decomposition of $\mathcal{C}$-grammars and since we will use it again later on, we give it a name.

[7] This means given a language in $\mathsf{Alg}(\mathcal{C})$, one can construct a Parikh equivalent language in $\mathcal{C}$.

Van Leeuwen decomposition Let $\mathcal{C}$ be a substitution closed language class and $G = (N, T, P, S)$ be a $\mathcal{C}$-grammar with $|N| \geqslant 2$. We choose an $A \in N \setminus \{S\}$ and define

$$G_A = (\{A\}, T \cup N \setminus \{A\}, P_A, A), \text{ where } P_A = \{B \to L \in P \mid B = A\}.$$

In other words, P_A contains all productions of G whose left-hand side is A. Consider the substitution $\sigma\colon (T \cup N)^* \to \mathcal{P}((T \cup N \setminus \{A\})^*)$ with $\sigma(A) = L(G_A)$ and $\sigma(x) = \{x\}$ for $x \in T \cup N \setminus \{A\}$. Let G' be the $\mathcal{C}$-grammar with

$$G' = (N \setminus \{A\}, T, P', S), \text{ where } P' = \{B \to \sigma(L) \mid B \to L \in P\}.$$

The two grammars G_A and G' together constitute a *van Leeuwen decomposition*. It is easy to see that $L(G') = L(G)$, which justifies the term 'decomposition'.

What makes the van Leeuwen decomposition useful is that G_A has just one nonterminal and G' has one less nonterminal than G. Hence, using induction, the decomposition allows us to concentrate on substitutions and grammars with one nonterminal.

We will use the fact that grammars preserve Parikh equivalence.

Lemma 2.6.9. *Let $\mathcal{C}$ be any language class. If G is a $\mathcal{C}$-grammar and $\bar{G}$ is obtained from G by replacing each right-hand side with a Parikh-equivalent language, then we have $\Psi(L(\bar{G})) = \Psi(L(G))$. In particular, if $\sigma, \tau\colon X^* \to \mathcal{P}(Y^*)$ are substitutions with $\Psi(\sigma(x)) = \Psi(\tau(x))$ for each $x \in X$ and $\Psi(L) = \Psi(K)$, then $\Psi(\sigma(L)) = \Psi(\tau(K))$.*

Lemma 2.6.10. *Suppose $\mathcal{C}$ is a substitution closed full semi-AFL. If G is a $\mathcal{C}$-grammar with one nonterminal, then $\Psi(L(G)) \in \Psi(\mathcal{C})$.*

Proof. Since $\mathcal{C}$ is closed under finite unions, we may assume that $G = (\{S\}, T, P, S)$ comprises just one production $S \to L$. Using simple rational transductions, one can obtain the languages $L_0 = L \cap T^*$ and

$$L_1 = \{uv \mid u, v \in (T \cup \{S\})^*,\ uSv \in L\}$$

from L, meaning $L_0, L_1 \in \mathcal{C}$. Observe that we have $\Psi(L(G)) = \Psi(\sigma(SL_1^*))$, where $\sigma\colon (T \cup \{S\})^* \to \mathcal{P}(T^*)$ is the substitution for which $\sigma(S) = L_0$ and with $\sigma(x) = \{x\}$ for $x \in T$. Since $\mathcal{C}$ is closed under substitution (and in particular under Kleene iteration, since every full semi-AFL contains a^*), this proves $\Psi(L(G)) \in \Psi(\mathcal{C})$. □

The proof of Theorem 2.6.8 is now a matter of a simple induction.

Proof of Theorem 2.6.8. Since the inclusion "$\supseteq$" is clear, we prove "$\subseteq$". We show by induction on the number n of nonterminals in a grammar G that if each right-hand side L of G satisfies $\Psi(L) \in \Psi(\mathcal{C})$, then $\Psi(L(G)) \in \Psi(\mathcal{C})$.

Suppose $n = 1$. According to Lemma 2.6.9, we may assume that the right-hand sides of G are in $\mathcal{C}$. By Lemma 2.6.10, we have $\Psi(L(G)) \in \Psi(\mathcal{C})$.

Suppose $n > 1$. Let G_A and G' be a van Leeuwen decomposition of G. By the already established case $n = 1$, we have $\Psi(L(G_A)) \in \Psi(\mathcal{C})$. Hence, by Lemma 2.6.9 and since $\mathcal{C}$ is substitution closed, each right-hand side L of G' satisfies $\Psi(L) \in \Psi(\mathcal{C})$. Since G' has $n - 1$ nonterminals, induction yields $\Psi(L(G)) = \Psi(L(G')) \in \Psi(\mathcal{C})$. □

2.7 A hierarchy of language classes

Based on the operators $\mathsf{SLI}(\cdot)$ and $\mathsf{Alg}(\cdot)$ from the previous sections, we introduce a hierarchy of language classes that will be a recurring theme in this work. Let F_0 be the class of finite languages and let

$$\mathsf{G}_i = \mathsf{Alg}(\mathsf{F}_i), \quad \mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{G}_i) \quad \text{for each } i \geqslant 0, \qquad \mathsf{F} = \bigcup_{i \geqslant 0} \mathsf{F}_i.$$

Then we clearly have the inclusions $\mathsf{F}_0 \subseteq \mathsf{G}_0 \subseteq \mathsf{F}_1 \subseteq \mathsf{G}_1 \subseteq \cdots$. Furthermore, G_0 is the class of context-free languages, F_1 is the smallest Presburger closed class containing CF, G_1 the algebraic extension of F_1, etc. In particular:

Theorem 2.7.1. F *is the smallest Presburger closed and algebraically closed language class containing the context-free languages.*

One might wonder why F_0 is not chosen to be the regular languages, which would have lead to the same class F. While this would be a natural choice, we want the following to hold. Note that the regular languages are not Presburger closed.

Proposition 2.7.2. *For each* $i \geqslant 0$*, the class* F_i *is an effective Presburger closed full semi-trio. Moreover, for each* $i \geqslant 0$*,* G_i *is an effective full semi-AFL. Furthermore, there is a uniform algorithm that, given* $i \in \mathbb{N}$*, a language in* F_i *(*G_i*), and one of the mentioned closure operators, computes the resulting language in* F_i *(*G_i*).*

Note that the third statement of the proposition means that not only each of the levels admits an algorithm for the closure operators but that there is one algorithm that can apply the closure operators for all levels (and it always yields a language on the same level as the input language). Proposition 2.7.2 follows from Proposition 2.5.1 and Proposition 2.6.1. The uniform algorithm recursively applies the transformations described in Proposition 2.5.1 and Proposition 2.6.1.

Proposition 2.7.3. *The class* F *is semilinear. Moreover, Parikh images of languages in* F *are effectively computable.*

The semilinearity follows from the fact that both $\mathsf{Alg}(\cdot)$ (Theorem 2.6.7) and $\mathsf{SLI}(\cdot)$ (Proposition 2.5.2) preserve semilinearity. Moreover, applying the procedure outlined in the remarks after Theorem 2.6.7 and the procedure in Proposition 2.5.2 yields a recursive algorithm to compute Parikh images of languages in the class F.

2.8 Well-quasi-orderings

In this work, we will often use well-quasi-orderings, which are a useful combinatorial tool for proving various finiteness conditions and structural properties.

A *quasi-ordering* on a set S is a binary reflexive transitive relation $\leqslant$. Let S be a set and $\leqslant$ be a quasi-order. A subset $T \subseteq S$ is *upward closed* if $s \in T$ and $s \leqslant t$ imply $t \in T$. It is called *downward closed* if $s \in T$ and $t \leqslant s$ imply $t \in T$. For a subset $T \subseteq S$, its *upward closure*, denoted $T{\uparrow}$, is the smallest upward closed set $T' \subseteq S$ containing T. The smallest downward closed set $T' \subseteq S$ containing T is called its *downward closure* and is denoted $T{\downarrow}$.

There are several common equivalent definitions of well-quasi-orderings, a concept which has been rediscovered several times; see **[Kruskal1972]** for an historical overview.

Theorem 2.8.1. *Let $\leqslant$ be a quasi-ordering on the set S. The following conditions are equivalent:*

1. *For every infinite sequence $s_1, s_2, \ldots$ of elements in S, there are indices $i < j$ such that $s_i \leqslant s_j$.*
2. *For every sequence $T_1 \subseteq T_2 \subseteq \cdots$ of upward closed subsets of S, there is an index n such that $T_m = T_n$ for $m \geqslant n$.*
3. *Every upward closed subset $T \subseteq S$ can be written as $T = \bigcup_{i=1}^{n} \{t_i\}\uparrow$ for some $t_1, \ldots, t_n \in T$.*
4. *Every non-empty subset $T \subseteq S$ has a finite non-empty set of minimal elements.*

A *well-quasi-ordering* is a quasi-order satisfying the equivalent conditions of Theorem 2.8.1. As a key ingredient in the so called well-structured transition systems of Finkel and Schnoebelen **[FinkelSchnoebelen2001]**, well-quasi-orderings are an important concept in the area of verification of infinite state systems. They have also been applied to the rational subset membership problem of wreath products by Lohrey, Steinberg, and the author of this work **[LohreySteinbergZetzsche2015a]**.

Here, we will employ the fact that two particular quasi-orderings are in fact well-quasi-orderings. Let X be an alphabet. For $\alpha, \beta \in X^{\oplus}$, we write $\alpha \leqslant \beta$ if $\alpha(x) \leqslant \beta(x)$ for all $x \in X$.

Theorem 2.8.2 (Dickson **[Dickson1913]**)**.** *For each alphabet X, the quasi-order $\leqslant$ on $X^{\oplus}$ is a well-quasi-ordering.*

The following lemma is a consequence of the fact that since $\leqslant$ is a well-quasi-order for multisets, downward closed sets can be represented by finitely many forbidden submultisets and are therefore recognizable.

Corollary 2.8.3. *For a given semilinear set $S \subseteq X^{\oplus}$, the set $\Psi^{-1}(S\downarrow)$ is an effectively computable regular language.*

Proof. Since S is Presburger-definable, the set $S' = X^{\oplus} \setminus (S\downarrow)$ is as well and hence effectively semilinear. Moreover, since $\leqslant$ is a well-quasi-ordering on $X^{\oplus}$, S' has a finite set F of minimal elements. Again, F is Presburger-definable because S' is and hence F is computable. Since S' is upward closed, we have $S' = F\uparrow$. Clearly, given $\mu \in X^{\oplus}$, the language $R_\mu = \{w \in X^* \mid \mu \leqslant \Psi(w)\}$ is an effectively computable regular language. Since $w \in \Psi^{-1}(S\downarrow)$ if and only if $w \notin \Psi^{-1}(F\uparrow)$, we have $X^* \setminus \Psi^{-1}(S\downarrow) = \bigcup_{\mu \in F} R_\mu$. Thus, we can compute a finite automaton for the complement, $\Psi^{-1}(S\downarrow)$. □

Higman1952 **[Higman1952]** and (apparently independently) **Haines1969** **[Haines1969]** have shown that the subword ordering on words is a well-quasi-ordering as well. For words $u, v \in X^*$, we write $u \preceq v$ if there are words $u_1, \ldots, u_n \in X^*$ and $v_0, \ldots, v_n \in X^*$ such that $u = u_1 \cdots u_n$ and $v = v_0 u_1 v_1 \cdots u_n v_n$. In this case, u is called a *subword* of v.

Theorem 2.8.4 (Higman **[Higman1952]** / Haines **[Haines1969]**)**.** *For each alphabet X, the quasi-order $\preceq$ on X^* is a well-quasi-ordering.*

Corollary 2.8.5. *If* $L \subseteq X^*$ *is upward closed or downward closed, then* L *is regular.*

Proof. Suppose L is upward closed. By condition 3 in Theorem 2.8.1, L is a finite union of sets of the form $\{w\}\uparrow$, which are easily seen to be regular. Furthermore, if L is downward closed, then its complement is upward closed and thus regular. □

This means in particular, that for any language $L \subseteq X^*$ whatsoever, the languages $L\uparrow$ and $L\downarrow$ are regular. In Chapter 9, we will study for which monoids M one can effectively compute a finite automaton for the downward closure of members of $\mathsf{VA}(M)$.

2.9 Conclusion

We have introduced the basic concepts that are needed for the remaining chapters. Aside from defining notation and recalling the notion of valence automata, we presented the new concept of graph monoids, which will be used throughout this work to characterize monoids with certain computational properties. As demonstrated already in this chapter, they accommodate a range of different storage mechanisms. Furthermore, we have defined a hierarchy of languages that will conveniently subdivide the languages accepted by a new class of storage mechanisms (stacked counters) in later chapters.

While introducing these fundamental concepts, this chapter also presented some technical contributions. These are mainly a description of the languages accepted by free products with amalgamation in terms of algebraic extensions (Theorem 2.6.3) and the result that taking the free product with two copies of $\mathbb{B}$ yields the whole algebraic extension (Theorem 2.6.6). Hence, the latter result complements the former and gives a precise account of the effect of *building stacks* on the expressive power.

The results in this chapter have appeared in **[Zetzsche2013a]** and in **[BuckheisterZetzsche2013a]**.

Related work Let us use this opportunity to survey some related work on the general concepts of this thesis.

Unifying automata frameworks Of course, there are many other frameworks that unify automata with storage. Very general frameworks are *Balloon Automata* of **HopcroftUllman1967** **[HopcroftUllman1967]**, *Abstract Families of Acceptors* of **GinsburgGreibach1967** **[GinsburgGreibach1967]**, and *Automata with Storage* of **Engelfriet2014** **[Engelfriet2014]**. These models roughly mimic what in the introduction is called storage mechanism: One has arbitrary partial functions on sets of states and has to arrive in a prescribed set of final states. This means, in particular, that they subsume valence automata. In fact, Greibach has obtained a result analogous to Theorem 2.6.6 in terms of Abstract Families of Acceptors **[Greibach1970]**.

However, as this work attempts to demonstrate, restricting attention to monoids, or further to graph monoids, allows for explanatory characterizations of computational properties. Consider, for example, the question of which storage mechanism admit a decision procedure for the emptiness problem. While there is no (complete) characterization of decidability of

the emptiness problem for graph monoids as of yet (see Section 4.3), a description of those graphs would certainly be illuminating. With models as general as the above, it is not clear whether a meaningful characterization, that does not essentially reformulate decidability, is possible at all.

The strength of these general frameworks seems to lie within their ability to capture general ideas that hold for arbitrary storage mechanisms. For example, **EngelfrietHoogeboom1993** [**EngelfrietHoogeboom1993**] have obtained connections between acceptance types of automata over infinite words that are independent of the storage mechanism.

A mathematical area that formalizes computing devices with a far broader scope than valence automata is that of *coalgebra* [**Rutten2000**]. A range of concepts of theoretical computer science has counterparts in coalgebra, and recent work of **GoncharovMiliusSilva2014** [**GoncharovMiliusSilva2014**] introduces the framework of $\mathbb{T}$-*automata*, which subsumes valence automata, albeit without ε-transitions. While ε-transitions can occur syntactically, a semantic of the corresponding coalgebra that mimics their behavior in valence automata is currently under development.

Another recent framework is that of *Auxiliary Storage with Bounded Tree-Width* of **MadhusudanParlato2011** [**MadhusudanParlato2011**]. Its purpose is, however, to explain a number of recent decidability results for emptiness problems. It is therefore tailored to guarantee decidability of emptiness rather than to cover a wide variety of mechanisms. It seems likely that it is incomparable in its modeling capacity to valence automata (see Section 7.4). Since **MadhusudanParlato2011** also obtain a Parikh's theorem, we compare not only our results on decidability of the emptiness problem with their framework, but also those on semilinearity. We therefore refer the reader to the conclusion sections 4.4 and 7.4.

A different model that also equips finite automata with an algebraic structure, namely semirings, is that of *Weighted Automata* [**DrosteKuichVogler2009**]. However, the semirings are not meant to represent storage mechanisms, but rather to specify a quantitative notion of behavior for the automaton.

Valence automata The research on valence automata conducted so far can be roughly divided into three directions:

- Investigating valence automata over monoids that either arise naturally in the theory of groups or semigroups or model a particular concrete storage mechanism. This has been done, for example, by **IbarraSahniKim1976** [**IbarraSahniKim1976**], **MitranaStiebe2001** [**MitranaStiebe2001**], **FernauStiebe2002a** [**FernauStiebe2002a**], **Corson2005** [**Corson2005**], **RenderKambites2009** [**RenderKambites2009**], **Kambites2009** [**Kambites2009**], **Render2010** [**Render2010**], and **Sorokin2014** [**Sorokin2014**].
- Studying the utility of valence automata to describe groups by accepting their identity languages. The type of questions is motivated by a celebrated theorem of Muller, Schupp, and Dunwoody [**Dunwoody1985**]. It characterizes those groups described by context-free grammars and can be stated as follows: The identity language of a group G is accepted by a valence automaton over some free group if and only if G is virtually free. Here, *virtually* means that G has a free subgroup of

finite index (see Section 6.3 for details). This raises the following question. For which classes $\mathcal{G}$ of groups is the following true: Every group G whose identity language is accepted by a valence automaton over some $H \in \mathcal{G}$, is itself virtually in $\mathcal{G}$ (meaning that G has a finite index subgroup contained in $\mathcal{G}$).

Representatives for this line of research are **Kambites2006** [**Kambites2006**], **ElstonOstheimer2004** [**ElstonOstheimer2004**], **ElderKambitesOstheimer2008** [**ElderKambitesOsthei**
GilmanShapiro1998 [**GilmanShapiro1998**], and **ClearyElderOstheimer2006** [**ClearyElderOstheimer2**
While the latter works sometimes assume determinism or a slightly different acceptance condition, they all use storage mechanisms defined by monoids.

- Using valence automata for decision procedures concerning groups or monoids. For example, valence automata have been used explicitly by **KambitesSilvaSteinberg2007** [**KambitesSilvaSteinberg2007**] and implicitly by **LohreySteinberg2008** [**LohreySteinberg2008**] to solve the rational subset membership problem for particular groups.

Graph monoids There are several notions of monoids similar to graph monoids. As described in Section 2.4, our graph monoids generalize the concept of graph groups [**Charney2007**]. Similar but different ways of defining monoids have been studied by **Silva2008** [**Silva2008**] and **Wrathall1991** [**Wrathall1991**]. **Silva2008** defines monoids by designating certain generators of a monoid to be invertible, some only one-sided invertible and some not invertible at all. However, his definition does not allow commutation among the generators. **Wrathall1991**, on the other hand, defines commutation by edges, but each of her generators is either invertible in both directions or not invertible at all.

Counters and semilinear intersection The idea that adding blind counters preserves semilinearity appears often in the literature. It is sometimes formulated for reversal-bounded counters, which are in most contexts equivalent to blind counters (see [**Greibach1978**]; for a translation that is economic in the number of counters, see [**JantzenKurganskyy2003**]). That reversal-bounded counters and blind counters alone guarantee semilinearity was shown by **Ibarra1978** [**Ibarra1978**] and **Greibach1978** [**Greibach1978**], respectively.

The preservation of semilinearity has been observed in a similar setting by **HarjuIbarraKarhumakiSalomaa2002** [**HarjuIbarraKarhumakiSalomaa2002**] and applied to language theoretic decision problems. Recently, this preservation was used by **LohreySteinberg2008** [**LohreySteinberg2008**] in a decision procedure for the rational subset membership of graph groups (see Theorem 4.3.9).

Building stacks and grammars A result that is analogous to our generalization of the equivalence between *building stacks* and grammars (Theorem 2.6.6) was obtained by **Greibach1970** [**Greibach1970**]. It should be mentioned that the transformation of building stacks is similar but crucially different from the way *Higher-Order Pushdown Automata* [**Damm1982**, **Engelfriet1991**] work. While our stacks only allow *push*, *pop* if empty, and a manipulation of the topmost entry, higher-order pushdowns can replicate their topmost

entry. This leads to very different computational properties. For example, building stacks in our sense preserves semilinearity, while higher-order pushdown automata lack semilinearity just above the level of ordinary pushdowns [**DammGoerdt1982**].

Acknowledgements I would like to thank Gretchen Ostheimer for a discussion about valence automata over groups and Sergey Goncharov and Stefan Milius for kindly answering questions about their work.

Chapter 3

Valence models vs. classical models

3.1 Introduction

Whenever we add a storage mechanism to a model of computation, one of the most fundamental questions is whether the extended model exhibits new behavior. In the context of monoid-defined storage mechanisms, this translates into the question of which monoids increase the possible behaviors of a model when employed as a storage mechanism.

In this work, we measure the expressiveness and behavior of valence automata and other models by the class of languages they induce. Therefore, the above question becomes: Which monoids cause the extended model to yield more languages than the model without the storage mechanism?

We study this question for valence automata and for valence grammars. The latter model extends context-free grammars in the same way valence automata extend finite automata: In a valence grammar, each production carries an element of a monoid. As in the case of valence automata, a derivation is valid if the product of the monoid elements (multiplied in the same order as the productions were applied) is the identity.

The concrete questions studied in this chapter therefore ask (i) for which monoids M can valence automata over M only accept regular languages and (ii) for which monoid monoids M can valence grammars over M generate only context-free languages. The main result of this chapter (Theorem 3.1.2) states that these conditions are equivalent and provides an algebraic characterization: It is shown that the conditions are satisfied if and only if $R_1(N)$ is finite for every finitely generated submonoid N of M. In the case of valence automata, the algebraic characterization has been obtained independently by **Render2010** in her PhD thesis [**Render2010**]. If the monoid at hand is a group, it is a simple consequence of an early result of **Anisimov1971** [**Anisimov1971**] (see Theorem 3.3.1) and has been observed by **MitranaStiebe2001** [**MitranaStiebe2001**].

While it is not hard to see that the monoids with finite sets $R_1(N)$ cause no increase of expressiveness in valence automata, this is non-trivial for valence grammars. This is due to the fact that in order to arrive at the monoid identity, one has to take into account the order of productions that are applied far apart in the

derivation tree. To illustrate the difficulty, we remark that with a slight variation in the definition, finite monoids produce far more than the context-free languages: Allowing finite monoids and *target sets* in valence grammars, meaning instead of the identity, one has to reach an element of a specified subset $T \subseteq M$, yields the class of matrix languages [**FernauStiebe2001**]. Furthermore, it was an open question by Fernau and Stiebe [**FernauStiebe2002a**] whether all languages generated by valence grammars over finite monoids are context-free and our result settles this question affirmatively.

In fact, it turns out that the conditions above are also equivalent to another behavioral property of valence automata: They are satisfied if and only if every valence automata over M has an equivalent that is deterministic in a strong sense. Here, we call a valence automaton deterministic if each of its moves is determined by the current state and input symbol. This again generalizes a result of **MitranaStiebe2001** [**MitranaStiebe2001**], who have shown that valence automata over a group cannot be determinized if the group contains at least one element of infinite order. Finally, we also observe that the conditions above also characterize those monoids over which valence transducers perform only rational transductions. Before we state the main result formally, we define the relevant notions.

Valence grammars Let M be a monoid. A *valence grammar over* M is a tuple $G = (N, T, M, P, S)$, where

- N, T are disjoint alphabets, called the *nonterminal* and *terminal alphabet*, respectively,
- P is a finite subset of $N \times (N \cup T)^* \times M$, called the set of *productions*, and
- $S \in N$ is the *start symbol*.

Instead of $(A, w, m) \in P$, we also write $(A \to w; m)$. The *derivation relation* is a binary relation on $(N \cup T)^* \times M$, for which

$$(u, a) \Rightarrow_G (u', a') \qquad \text{if } u = rAs,\ u' = rws,\ a' = am \text{ for some } (A \to w; m) \in P.$$

The *language generated by* G is then

$$L(G) = \{w \in T^* \mid (S, 1) \Rightarrow_G^* (w, 1)\}.$$

The class of languages generated by valence grammars over M is denoted by $VG(M)$.

Valence grammars were introduced by **Paun1980** [**Paun1980**]. A thorough treatment, including normal form results and a classification of the resulting language classes for commutative monoids, has been carried out by **FernauStiebe2002a** [**FernauStiebe200**

Example 3.1.1. *Let* $n \in \mathbb{N}$ *and let* $G = (N, T, \mathbb{Z}^n, P, S)$ *be the valence grammar with* $N = \{S\}$, $T = \{a_i, \bar{a}_i \mid 1 \leqslant i \leqslant n\}$, *and where* P *consist of the productions*

$$\begin{aligned} &(S \to S; && (1, \dots, 1)), \\ &(S \to Sa_iS\bar{a}_iS; && (0, \dots, 0, -1, 0, \dots, 0)), && \text{for } 1 \leqslant i \leqslant n, \\ &(S \to \varepsilon; && (0, \dots, 0)), \end{aligned}$$

where in the last production, the -1 is in the i-th component. Then the first production allows adding $(k, \dots, k)$ to the storage for $k \in \mathbb{N}$. Whenever the second production is applied, we decrement the i-th component. Hence, we arrive at $(0, \dots, 0)$ if and only if the second production has been applied the same number of times for each i (and the first production has also been applied this often). Moreover, the context-free productions guarantee that every generated word is in D_n, the semi-Dyck language. Therefore,

$$L(G) = \{w \in D_n \mid |w|_{a_i} = |w|_{a_j} \text{ for all } i, j \in \{1, \dots, n\}\}.$$

Deterministic valence automata A valence automaton $A = (Q, X, M, E, q_0, F)$ is called *deterministic* if

$$E \subseteq Q \times X \times M \times Q,$$

(meaning that in each step, precisely one input symbol is read) and for each pair $(q, x) \in Q \times X$, there is at most one edge $(q, x, m, q') \in E$ for $m \in M$ and $q' \in Q$. The class of languages accepted by deterministic valence automata over M is denoted by $\mathsf{detVA}(M)$.

Note that sometimes, valence automata over M are equivalent (with respect to accepted languages) to some automata model, but deterministic valence automata do not correspond to the usual deterministic variant of the model. This is because in these deterministic variants, a step may depend on the storage content. In our case, each step must be determined by the current state and input, necessitating in particular the requirement that every edge read input. For example, while valence automata over $\mathbb{B}^{(2)}$ are equivalent to pushdown automata, deterministic valence automata over $\mathbb{B}^{(2)}$ are less powerful than deterministic pushdown automata: The language $L = a^*c \cup \{a^n b^n \mid n \geqslant 0\}$ is deterministic context-free (and even accepted by the weak variant that accepts with empty stack and final state [**AutebertBerstelBoasson1997**]). However, it is easy to see that L is not in $\mathsf{detVA}(\mathbb{B}^{(2)})$: After reading a^n, a deterministic valence automaton would have to be in a configuration from which c leads to a final configuration. Hence, if a^m and a^n lead to the same state, they already lead to the same configuration, which is clearly a contradiction.

Valence transducers Let M be a monoid. A *valence transducer* is an automaton over $X^* \times M \times Y^*$, where X and Y are alphabets. As an automaton $X^* \times M \times Y^*$, a valence transducer has a step relation $\to_A$ on $Q \times X^* \times M \times Y^*$, which allows us to define

$$T(A) = \{(u, v) \in X^* \times Y^* \mid (q_0, \varepsilon, 1, \varepsilon) \to_A^* (q, u, 1, v) \text{ for some } q \in F\}.$$

$T(A)$ is called the *transduction performed by* A. Intuitively, a valence transducer over M is a valence automaton over M where each edge also carries an output word. The class of transductions performed by valence transducers over M is denoted by $\mathsf{VT}(M)$.

We are now ready to state the main result of this chapter.

Theorem 3.1.2. *Let M be a monoid. The following conditions are equivalent:*

1. $\mathsf{VA}(M)$ *contains only regular languages.*
2. $\mathsf{VG}(M)$ *contains only context-free languages.*

3. $\mathsf{VT}(M)$ *contains only rational transductions.*

4. $\mathsf{detVA}(M) = \mathsf{VA}(M)$.

5. $\mathsf{R}_1(N)$ *is finite for every finitely generated submonoid* N *of* M.

The rest of this chapter devoted to the proof of Theorem 3.1.2. In Section 3.2, we prove an algebraic dichotomy of monoids: It is shown that each monoid satisfies precisely one of two conditions that can be employed for language theoretic arguments. In Section 3.3, we show the equivalence among conditions 1 and 3 to 5. Section 3.4 then completes the proof of Theorem 3.1.2 by establishing the equivalence between condition 2 and condition 5.

The results of this chapter have appeared in **[Zetzsche2011b]**.

3.2 A dichotomy of monoids

In this section, we prove a dichotomy of monoids (Corollary 3.2.3). Its two cases will be exploited in Sections 3.3 and 3.4 to deduce language theoretic properties. After submission of **[Zetzsche2011b]**, the author learned that this dichotomy had been well known to semigroup theorists (see Section 3.5 for details).

Lemma 3.2.1. *Let* $r, s \in M$ *with* $rs = 1$ *and* $sr \neq 1$. *Then* $\langle r, s\rangle$ *is isomorphic to the bicyclic monoid.*

Proof. First, we claim that $r^k = r^\ell$ implies $k = \ell$. Suppose $r^k = r^\ell$ for $k < \ell$. Then

$$sr = r^k s^k sr = r^\ell s^k sr = r^{\ell-k} sr = r^{\ell-k-1} r = r^{\ell-k} = r^\ell s^k = r^k s^k = 1,$$

a contradiction proving the claim. Furthermore, $s^k r^k = 1$ implies $k = 0$. Indeed, if $s^k r^k = 1$ with $k > 0$, then

$$sr = s^k r^k sr = s^k r^{k-1} r = s^k r^k = 1.$$

Suppose $s^k r^\ell = s^m r^n$. Without losing generality, we assume $k \leqslant m$. Multiplying r^m from the left yields $r^{m-k+\ell} = r^n$ and hence $m - k + \ell = n$. This means $n - \ell = m - k \geqslant 0$. Therefore

$$s^{m-k} r^{n-\ell} = r^k (s^m r^n) s^\ell = r^k (s^k r^\ell) s^\ell = 1$$

and thus $m = k$ and $n = \ell$.

Let $X = \{x, \bar{x}\}$. The morphism $\varphi\colon X^* \to \langle r, s\rangle$ with $\varphi(x) = r$ and $\varphi(\bar{x}) = s$ satisfies $\varphi(x\bar{x}) = 1$ and can therefore be lifted to a morphism $\hat{\varphi}\colon \mathbb{B} \to \langle r, s\rangle$. It is clearly surjective and and an equality $\hat{\varphi}([\bar{x}]^k [x]^\ell) = \hat{\varphi}([\bar{x}]^m [x]^n)$ implies $s^k r^\ell = s^m r^n$, meaning $k = m$ and $\ell = n$ as shown above. Thus, $\hat{\varphi}$ is also injective. □

The following proposition is a first dichotomy that we will use to derive Corollary 3.2.3.

Proposition 3.2.2. *For each monoid* M, *exactly one of the following holds:*

1. $\mathsf{J}_1(M)$ *is a subgroup of* M.

2. M *contains a copy of the bicyclic monoid as a submonoid.*

Proof. If M contains a copy of the bicyclic monoid as a submonoid, this copy is included in $J_1(M)$, meaning $J_1(M)$ contains elements x and y with $xy = 1$ and $yx \neq 1$, which cannot happen in a group. Thus, the two conditions are mutually exclusive.

Suppose M does not contain a copy of the bicyclic monoid. By Lemma 3.2.1, this means whenever $rs = 1$, we also have $sr = 1$. For $a \in J_1(M)$, there are $x, y \in M$ with $xay = 1$. Therefore, we also have $ayx = 1$ and $yxa = 1$. This implies that yx is a two-sided inverse of a and lies in $J_1(M)$.

The subset $J_1(M)$ is also closed under composition. Indeed, let $a, b \in J_1(M)$ and let s be a two-sided inverse of a and t be a two-sided inverse of b. Then $sabt = 1$ and hence $ab \in J_1(M)$. □

In order to formulate our dichotomy, we need to define some notation. For each $x \in R_1(M)$ and $x' \in L_1(M)$, we define

$$\overrightarrow{I}(x) = \{y \in M \mid xy = 1\}, \qquad \overleftarrow{I}(x') = \{y' \in M \mid y'x' = 1\}.$$

The elements in $\overrightarrow{I}(x)$ are called *right inverses* of x and the elements of $\overleftarrow{I}(x')$ are called *left inverses* of x'.

Corollary 3.2.3. *For each monoid M, exactly one of the following holds:*

1. *The subsets $R_1(M)$, $L_1(M)$, and $J_1(M)$ coincide and constitute a finite group.*

2. *There are infinite sets $S \subseteq R_1(M)$ and $S' \subseteq L_1(M)$ such that $\overrightarrow{I}(s) \cap \overrightarrow{I}(t) = \emptyset$ for $s, t \in S$, $s \neq t$, and $\overleftarrow{I}(s') \cap \overleftarrow{I}(t') = \emptyset$ for $s', t' \in S'$, $s' \neq t'$.*

Proof. If M contains a copy $\langle r, s \rangle$ of the bicyclic monoid as a submonoid, the infinite sets $\{r^i \mid i \geqslant 0\} \subseteq R_1(M)$ and $\{s^i \mid i \geqslant 0\} \subseteq L_1(M)$ satisfy our second condition. Indeed, if $x \in \overrightarrow{I}(r^i) \cap \overrightarrow{I}(r^j)$ for $i \leqslant j$, then $r^{j-i} = r^{j-i}r^ix = r^jx = 1$ and hence $i = j$. Similarly, we can show that $\overleftarrow{I}(s^i) \cap \overleftarrow{I}(s^j) = \emptyset$ for $i \neq j$.

Otherwise, by Proposition 3.2.2, $J_1(M)$ is a group, meaning that the three subsets $R_1(M)$, $L_1(M)$, and $J_1(M)$ coincide. If $J_1(M)$ is finite, M satisfies our first condition. If $J_1(M)$ is infinite, we can choose $S = S' = J_1(M)$ as infinite sets for our second condition. □

3.3 Valence automata vs. finite automata

In this section, we show that the following conditions are equivalent:

1. $\mathsf{VA}(M)$ contains only regular languages.

2. $\mathsf{VT}(M)$ contains only rational transductions.

3. $\mathsf{detVA}(M) = \mathsf{VA}(M)$.

4. $R_1(N)$ is finite for every finitely generated submonoid N of M.

The equivalence of the first and the last condition has been obtained independently by Render **[Render2010]**.

Note that $\mathsf{VA}(M)$ is included in the regular languages if and only if every identity language of M is regular (see Theorem 2.3.3). The class of finitely generated groups for which each identity language is regular was understood early on. The following is easy to see (and follows from Lemmas 3.3.3 and 3.3.4).

Theorem 3.3.1 (Anisimov **[Anisimov1971]**)**.** *Let G be a finitely generated group. Then every identity languages of G is regular if and only if G is finite.*

Translated to our setting, this means that for a finitely generated group G, $\mathsf{VA}(G)$ contains only regular languages if and only if G is finite. This has also been observed by **MitranaStiebe2001 [MitranaStiebe2001]**.

Lemma 3.3.2. *For each monoid M, the following conditions are equivalent:*

1. $\mathsf{VA}(M)$ *contains only regular languages.*
2. $\mathsf{VT}(M)$ *contains only rational transductions.*

Proof. Since $\mathsf{VT}(M)$ contains precisely those transductions that are homomorphic images of languages in $\mathsf{VA}(M)$, the first condition clearly implies the second. On the other hand, if $\mathsf{VA}(M)$ contains a non-regular language L, the transduction $\{\varepsilon\} \times L$ is contained in $\mathsf{VT}(M)$ and is clearly not rational. □

The following is a simple consequence of Corollary 3.2.3.

Lemma 3.3.3. *Let $\mathsf{R}_1(N)$ be finite for every finitely generated submonoid N of M. Then, all languages in $\mathsf{VA}(M)$ are regular.*

Proof. Let $A = (Q, X, M, E, q_0, F)$ be a valence automaton over M. Since E is finite, the set of $m \in M$ for which there is some edge $(p, (w, m), q)$ in E is finite. If N is the submonoid of M generated by these $m \in M$, we can regard A as a valence automaton over N. Thus, let $A = (Q, X, N, E, q_0, F)$.

Removing edges of the form $(p, (w, m), q)$ such that $m \notin \mathsf{J}_1(N)$ will not alter the accepted language, since such edges cannot be used in a successful run. According to Corollary 3.2.3, $\mathsf{J}_1(N)$ is a finite group and we may assume $A = (Q, X, \mathsf{J}_1(N), E, q_0, F)$. Since $\mathsf{J}_1(N)$ is finite, a finite automaton accepting $L(A)$ can easily be constructed by incorporating the monoid elements into the states. □

MitranaStiebe2001 [MitranaStiebe2001] have shown that valence automata over groups with at least one element of infinite order cannot be determinized. We can use a similar idea and the dichotomy of monoids to provide a characterization of those monoids over which valence automata can be determinized.

Lemma 3.3.4. *Let M contain a finitely generated submonoid N such that $\mathsf{R}_1(N)$ is infinite. Then, $\mathsf{detVA}(M)$ is strictly included in $\mathsf{VA}(M)$. In particular, $\mathsf{VA}(M)$ contains a non-regular language.*

Proof. Let N be generated by the finite set $\{a_1, \ldots, a_n\}$ and let $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$ be disjoint alphabets. Let $\varphi\colon (X \cup Y)^* \to N$ be the epimorphism defined by $\varphi(x_i) = \varphi(y_i) = a_i$ and $K = X^* \cup \{w \in X^*Y^* \mid \varphi(w) = 1\}$. Then, K is clearly contained in $\mathsf{VA}(M)$.

Suppose K were accepted by a deterministic valence automaton A over M. Let $S \subseteq R_1(N)$ be the infinite set provided by Corollary 3.2.3. The infinity of S implies that we can find an infinite set $T \subseteq X^*$ such that $\varphi(T) = S$ and $\varphi(u) \neq \varphi(v)$ for $u, v \in T$ with $u \neq v$. Since A is deterministic and $T \subseteq L(A)$, each word $w \in T$ causes A to enter a configuration $(q(w), 1)$ where $q(w)$ is a final state.

Choose $u, v \in T$ such that $u \neq v$ and $q(u) = q(v)$. Find a word $u' \in Y^*$ such that $\varphi(u)\varphi(u') = 1$. This is possible since $\varphi(u) \in R_1(N)$ and φ is surjective when restricted to Y^*. The word u' causes A to go from $(q(u), 1) = (q(v), 1)$ to $(q, 1)$ for some final state q, since $uu' \in K$. Thus, vu' is also contained in K and hence $\varphi(v)\varphi(u') = 1$, but $\overrightarrow{1}_N(\varphi(u)) \cap \overrightarrow{1}_N(\varphi(v)) = \emptyset$, a contradiction. Hence, K is not contained in $\mathsf{detVA}(M)$. □

3.4 Valence grammars vs. context-free grammars

In this section, it is shown that the following conditions are equivalent:

1. $\mathsf{VG}(M)$ contains only context-free languages.
2. $R_1(N)$ is finite for every finitely generated submonoid N of M.

In one of the directions, we have to construct a context-free grammar for valence grammars over monoids that fulfill the second condition. Because of the limited means available in the context-free case, the constructed grammar can simulate only a certain fragment of the derivations in the valence grammar. Thus, we will have to make sure that every word generated by the valence grammar has a derivation in the aforementioned fragment. These derivations are obtained by considering the derivation tree of a given derivation and then choosing a suitable linear extension of the tree order. The construction of these linear extensions can already be described for a simpler kind of partial order, valence trees.

Valence trees and excursiveness Let X be an alphabet and $U \subseteq X$ a subset. Then, each word $w \in X^*$ has a unique decomposition $w = y_0x_1y_1 \cdots x_ny_n$ such that $y_0, y_n \in (X \setminus U)^*$, $y_i \in (X \setminus U)^+$ for $1 \leqslant i \leqslant n-1$, and $x_i \in U^+$ for $1 \leqslant i \leqslant n$. This decomposition is called the U-*decomposition* of w and we define $\rho(w, U) = n$.

A *tree* is a finite partially ordered set $(\mathcal{T}, \leqslant)$ that has a least element and where, for each $t \in \mathcal{T}$, the set $\{t' \in \mathcal{T} \mid t' \leqslant t\}$ is totally ordered by $\leqslant$. The least element is called the *root* and the maximal elements are called *leaves*. A *valence tree* $\mathcal{T}$ over M is a tuple $(\mathcal{T}, \leqslant, \varphi)$, where $(\mathcal{T}, \leqslant)$ is a tree and $\varphi\colon \mathcal{T}^* \to M$ is a morphism[1] assigning a *valence* to each node. An *evaluation* defines an order in which the nodes in a valence tree can be traversed that is compatible with the tree order. Thus, an *evaluation* of $\mathcal{T}$ is a linear extension $\preceq$ of $(\mathcal{T}, \leqslant)$. Let $w \in \mathcal{T}^*$ correspond to $\preceq$, i.e., let $\mathcal{T} = \{t_1, \ldots, t_n\}$ such that $t_1 \preceq \cdots \preceq t_n$ and $w = t_1 \cdots t_n$. Then, the *value* of $\preceq$ is defined to be $\varphi(w)$. An element $v \in M$ is called a *value of* $\mathcal{T}$ if there exists an evaluation of $(\mathcal{T}, \leqslant)$ with value v. Given a node $t \in \mathcal{T}$, let $U_t = \{t' \in \mathcal{T} \mid t \leqslant t'\}$. If $w = y_0x_1y_1 \cdots x_ny_n$ is the U_t-decomposition of w, then $\varphi(x_1), \ldots, \varphi(x_n)$ is called the *valence sequence* of t in w and n its *length*. By the *excursiveness* of an evaluation, we refer to the maximal length of a valence sequence. Hence, the excursiveness of an evaluation is the maximal number of

[1] We will often assume, without loss of generality, that $\mathcal{T}$ is an alphabet.

times one has to enter any given subtree when traversing the nodes in the order given by the evaluation.

The simulated fragment of the derivations of the valence grammar consists of those derivations whose derivation tree admits an evaluation of bounded excursiveness. Hence, proving completeness of this fragment amounts to finding evaluations of valence trees with small excursiveness.

Of course, for every valence tree, there are evaluations with excursiveness one (take, for example, the order induced by a preorder traversal), but these might not be able to exhaust all possible values. However, we will see in Lemma 3.4.3 that, in the case of a finite group, there exists a bound m such that every value can be attained by an evaluation of excursiveness of at most m. Combining this with the fact that a restriction to finite groups means no loss of generality will then complete the proof.

In order to show Lemma 3.4.3, we need two combinatorial facts. The first one will allow us to conclude that certain rearrangements of an evaluation do not alter its value.

Lemma 3.4.1. *For each finite group G, there is a constant $m \in \mathbb{N}$ with the following property: For elements $g_i, h_i \in G$, $i = 1, \dots, n$, $n \geqslant m$, there are indices $k, \ell \in \mathbb{N}$, $1 \leqslant k < \ell \leqslant n$, such that*

$$g_k h_k g_{k+1} h_{k+1} \cdots g_\ell h_\ell = g_k g_{k+1} \cdots g_\ell h_k h_{k+1} \cdots h_\ell.$$

Proof. Let $m = 2(|G|^3 + 1)$ and $D \subseteq \{1, \dots, n\}$ be the set of odd indices. Define the map $\alpha \colon D \to G^3$ by $\alpha(i) = (g_1 \cdots g_i,\ h_1 \cdots h_i,\ g_1 h_1 \cdots g_i h_i)$ for $i \in D$. Since $|D| \geqslant |G|^3 + 1$, there are indices $i, j \in D$, $i < j$, such that $\alpha(i) = \alpha(j)$. This means that $g_{i+1} \cdots g_j = 1$, $h_{i+1} \cdots h_j = 1$, and $g_{i+1} h_{i+1} \cdots g_j h_j = 1$. Since i, j are both odd, letting $k = i + 1$ and $\ell = j$ implies $k < \ell$ and yields the desired equality. □

The next lemma guarantees that certain rearrangements nowhere increase the length of valence sequences.

Lemma 3.4.2. *Let X be an alphabet and $U, V \subseteq X$ subsets such that either $U \subseteq V$, $V \subseteq U$, or $U \cap V = \emptyset$. Furthermore, let $r \in X^* U$, $x \in U^+$, $y \in (X \setminus U)^+$, and $s \in X^* \setminus UX^*$. Then, we have $\rho(rxys, V) \leqslant \rho(ryxs, V)$.*

Proof. Suppose $V \subseteq U$. Since y does not contain any symbols in V, we have

$$\begin{aligned} \rho(ryxs, V) &= \rho(r, V) + \rho(x, V) + \rho(s, V), \\ \rho(rxys, V) &= \rho(rx, V) + \rho(s, V). \end{aligned}$$

Thus,

$$\begin{aligned} \rho(rxys, V) &= \rho(rx, V) + \rho(s, V) \\ &\leqslant \rho(r, V) + \rho(x, V) + \rho(s, V) \\ &= \rho(ryxs, V). \end{aligned}$$

In the case $U \cap V = \emptyset$, x does not contain any symbol in V. Hence,

$$\begin{aligned} \rho(ryxs, V) &= \rho(r, V) + \rho(y, V) + \rho(s, V), \\ \rho(rxys, V) &= \rho(r, V) + \rho(ys, V), \end{aligned}$$

which implies

$$\begin{aligned}\rho(rxys, V) &= \rho(r, V) + \rho(ys, V)\\ &\leqslant \rho(r, V) + \rho(y, V) + \rho(s, V)\\ &= \rho(ryxs, V).\end{aligned}$$

Now suppose $U \subseteq V$. Since the rightmost letter of r is in V and x lies in V^+, we have $\rho(rxys, V) = \rho(rys, V)$. Thus, $\rho(rxys, V) = \rho(rys, V) \leqslant \rho(ryxs, V)$. □

The following states that the simulated fragment of derivations, namely those with bounded excursiveness, is in fact complete.

Lemma 3.4.3. *For each finite group* G*, there is a constant* m *such that every value of a valence tree over* G *has an evaluation of excursiveness of at most* m*.*

Proof. To each evaluation w of $(\mathcal{T}, \leqslant)$, we assign the multiset $\mu_w \in \mathcal{T}^{\oplus}$ that is defined by $\mu_w(t) = \rho(w, U_t)$ for every $t \in \mathcal{T}$. That is, $\mu_w(t)$ is the length of the valence sequence of t in w.

Let m be the constant provided by Lemma 3.4.1 and let $w \in \mathcal{T}^*$ be an evaluation of $(\mathcal{T}, \leqslant)$ such that μ_w is minimal with respect to $\leqslant$ among all evaluations with the value v. If we can prove that $\mu_w(t) \leqslant m$ for all $t \in \mathcal{T}$, the lemma follows. Therefore, suppose that there is a $t \in \mathcal{T}$ with $n = \mu_w(t) > m$. Specifically, let $w = y_0x_1y_1 \cdots x_ny_n$ be the U_t-decomposition of w. Use Lemma 3.4.1 to find indices $1 \leqslant k < \ell \leqslant n$ with

$$\varphi(x_k)\varphi(y_k)\cdots\varphi(x_\ell)\varphi(y_\ell) = \varphi(x_k)\cdots\varphi(x_\ell)\varphi(y_k)\cdots\varphi(y_\ell). \tag{3.1}$$

Furthermore, let

$$w' = (y_0x_1y_1\cdots x_{k-1}y_{k-1})(x_k\cdots x_\ell y_k\cdots y_\ell)(x_{\ell+1}y_{\ell+1}\cdots x_ny_n). \tag{3.2}$$

That is, we obtain w' from w by replacing $x_ky_k\cdots x_\ell y_\ell$ with $x_k\cdots x_\ell y_k\cdots y_\ell$. Then, (3.1) means that $\varphi(w') = \varphi(w)$. We shall prove that w' is an evaluation of $(\mathcal{T}, \leqslant)$ and obeys $\mu_{w'} \leqslant \mu_w$ and $\mu_{w'} \neq \mu_w$, which contradicts the choice of w.

First, we prove that w' is an evaluation. Let $u_1, u_2 \in \mathcal{T}$ be nodes with $u_1 \leqslant u_2$. If $u_1 < t$, then u_1 appears in y_0, and thus u_2 is on the right side of u_1 in w'. If $u_1 \geqslant t$, then each of the nodes u_1, u_2 appears in some x_i and therefore do not change their relative positions. If u_1 and t are incomparable, then u_2 and t are also incomparable and each of u_1, u_2 appears in some y_i. Again, u_1 and u_2 do not change their relative positions. Thus, w' corresponds to a linear extension of the order $\leqslant$.

We want to show that $\mu_{w'} \leqslant \mu_w$. To this end, we consider the words

$$w_i = (y_0x_1y_1\cdots x_{k-1}y_{k-1})(x_k\cdots x_{k+i}y_k\cdots y_{k+i})(x_{k+i+1}y_{k+i+1}\cdots x_ny_n)$$

for $0 \leqslant i \leqslant \ell - k$. With these, we have $w = w_0$ and $w' = w_{\ell-k}$. Since $(\mathcal{T}, \leqslant)$ is a tree, we have $U_u \subseteq U_t$, $U_t \subseteq U_u$, or $U_u \cap U_t = \emptyset$ for every $u \in \mathcal{T}$. Therefore, we can apply Lemma 3.4.2 to $U = U_t$, $V = U_u$, and

$$\begin{aligned} r &= (y_0x_1y_1\cdots x_{k-1}y_{k-1})(x_k\cdots x_{k+i}), & x &= x_{k+i+1},\\ y &= y_k\cdots y_{k+i}, & s &= y_{k+i+1}(x_{k+i+2}y_{k+i+2}\cdots x_ny_n), \end{aligned}$$

which yields $\rho(w_{i+1}, U_u) \leqslant \rho(w_i, U_u)$ for $0 \leqslant i < \ell - k$. This implies the inequality $\mu_{w'}(u) \leqslant \mu_w(u)$ and therefore $\mu_{w'} \leqslant \mu_w$.

It remains to be shown that $\mu_{w'} \neq \mu_w$. In w', the node t has the valence sequence

$$\varphi(x_1), \ldots, \varphi(x_{k-1}), \varphi(x_k \cdots x_\ell), \varphi(x_{\ell+1}), \cdots \varphi(x_n),$$

which has length $\mu_{w'}(t) = n - (\ell - k) < n = \mu_w(t)$. □

Our next step is to apply our knowledge on valence trees to derivation trees of valence grammars.

Derivation trees of valence grammars A *derivation tree* for a valence grammar $G = (N, T, M, P, S)$ is a tuple $(\mathcal{T}, \leqslant, \varphi, (\leqslant_t)_{t \in \mathcal{T}}, \Lambda)$, where

- $(\mathcal{T}, \leqslant, \varphi)$ is a valence tree,
- for each $t \in \mathcal{T}$, $\leqslant_t$ is a total order on the set of successors of t,
- $\Lambda \colon \mathcal{T} \to N \cup T \cup \{\varepsilon\}$ defines a *label* for each node,
- if $t \in \mathcal{T}$ is a node with the successors $s_1, \ldots, s_n$ such that $s_1 \leqslant_t \ldots \leqslant_t s_n$, then we either have $\Lambda(t) \in T \cup \{\varepsilon\}$, $n = 0$, and $\varphi(t) = 1$ or we have $\Lambda(t) \in N$ and there is a production $(\Lambda(t) \to \Lambda(s_1) \cdots \Lambda(s_n); \varphi(t))$ in P.

The total orders $\leqslant_t$, $t \in \mathcal{T}$, induce a total order on the set of leaves, which in turn defines a word $w \in T^*$. This word is called the *yield* of the derivation tree.

Each derivation tree can be regarded as a valence tree. An evaluation then defines a derivation $(A, 1) \Rightarrow_G^* (w, v)$, where $A \in N$ is the label of the root, w is the yield, and $v \in M$ is the value of the evaluation. Conversely, every derivation induces a derivation tree and an evaluation. Thus, a word $w \in T^*$ is in $L(G)$ if and only if there exists a derivation tree for G with yield w, a root labeled S, and an evaluation with value 1.

Lemma 3.4.4. *Let $R_1(N)$ be finite for every finitely generated submonoid N of M. Furthermore, let $G = (N, T, M, P, S)$ be a valence grammar over M. Then, $L(G)$ is context-free.*

Proof. We can assume that M is finitely generated and thus has a finite $R_1(M)$. Since productions $(A \to w; m)$ with $m \notin J_1(M)$ cannot be part of a successful derivation, their removal does not change the generated language. Furthermore, by Corollary 3.2.3, $J_1(M)$ is a finite group. Thus, we can assume that $G = (N, T, H, P, S)$, where $H = J_1(M)$ is a finite group. By a simple construction, we can further assume that in G, every production is of the form $(A \to w; h)$ with $w \in N^*$ or $(A \to w; 1)$ with $w \in T \cup \{\varepsilon\}$.

We shall construct a context-free grammar $G' = (N', T, P', S')$ for $L(G)$. The basic idea is that G' simulates derivations of bounded excursiveness. This is done by letting the nonterminals in G' consist of a nonterminal $A \in N$ and a finite sequence σ of elements from H. G' then simulates the generation of a nonterminal A by generating a pair (A, σ) and thereby guesses that the corresponding node in the derivation tree of G will have σ as its valence sequence. Lemma 3.4.3 will then guarantee that this allows G' to derive all words in $L(G)$ when the sequences σ are of bounded length.

Formally, we regard H as an alphabet and a sequence is a word over H. In order to distinguish between the concatenation of words in H^* and the group operation in H, we will denote the concatenation in H^* by $\square$. Let $N' = N \times H^{\leqslant m}$, in which $m \in \mathbb{N}$ is the constant provided by Lemma 3.4.3 for the group H. The set of sequences that can be obtained from another sequence σ by "joining" subsequences is denoted by $J(\sigma)$:

$$J(h_1 \square h_2 \square \sigma) = J((h_1 h_2) \square \sigma) \cup \{h_1 \square \sigma' \mid \sigma' \in J(h_2 \square \sigma)\}$$

for $h_1, h_2 \in H$ and $\sigma \in H^*$ and $J(\sigma) = \{\sigma\}$ if $|\sigma| \leqslant 1$. J is defined for subsets $S \subseteq H^*$ by $J(S) = \bigcup_{\sigma \in S} J(\sigma)$.

For each production $(A \to w; h) \in P$, $w = B_1 \cdots B_n$, $B_i \in N$ for $1 \leqslant i \leqslant n$, we include the production

$$(A, \sigma) \to (B_1, \sigma_1) \cdots (B_n, \sigma_n),$$

for each $\sigma \in H^{\leqslant m} \setminus \{\varepsilon\}$ and $\sigma_1, \ldots, \sigma_n \in H^{\leqslant m}$ such that for $\sigma = h_1 \square \sigma'$, $h_1 \in H$, $\sigma' \in H^{\leqslant m-1}$, one of the following holds:

1. $(h^{-1} h_1) \square \sigma' \in J(\sigma_1 \sqcup \cdots \sqcup \sigma_n)$.
2. $h_1 = h$ and $\sigma' \in J(\sigma_1 \sqcup \cdots \sqcup \sigma_n)$.

Furthermore, for every production $(A \to w; 1)$ with $w \in T \cup \{\varepsilon\}$, we include the production $(A, \varepsilon) \to w$. Finally, the start symbol of G' is $(S, 1)$.

It remains to be shown that $L(G') = L(G)$. In order to prove $L(G') \subseteq L(G)$, one can show by induction on n that for $w \in T^*$, $(A, \sigma) \Rightarrow_{G'}^n w$ implies that there is a derivation $(A, 1) \Rightarrow_G^* (w, h)$ for some $h \in H$ using productions

$$(A_1 \to w_1; h_1), \ldots, (A_k \to w_k; h_k)$$

such that $\sigma \in J(h_1 \square \cdots \square h_k)$ (and of course $h = h_1 \cdots h_k$). This implies that for $(S, 1) \Rightarrow_{G'}^* w$, $w \in T^*$, we have $w \in L(G)$. Thus, $L(G') \subseteq L(G)$.

Let $w \in L(G)$ with derivation tree $(\mathcal{T}, \leqslant, \varphi, (\leqslant_t)_{t \in \mathcal{T}}, \Lambda)$. By Lemma 3.4.3, there is an evaluation $\preceq$ of the tree with value 1 and of excursiveness $\leqslant m$. From the tree and the evaluation, we construct a derivation tree $(\mathcal{T}, \leqslant, \varphi', (\leqslant_t)_{t \in \mathcal{T}}, \Lambda')$ for w in G' as follows. The components $\mathcal{T}$, $\leqslant$, and $\leqslant_t$, $t \in \mathcal{T}$, remain unaltered, but φ' will assign 1 to each node and Λ' is defined by $\Lambda'(t) = \Lambda(t)$ if $\Lambda(t) \in T \cup \{\varepsilon\}$ and $\Lambda'(t) = (\Lambda(t), h_1 \square \cdots \square h_k)$ if $\Lambda(t) \in N$, where $h_1, \ldots, h_k$ is the valence sequence of t in $\preceq$. Now, one can see that the new tree is a derivation tree for G' that generates w. Hence, $L(G) \subseteq L(G')$. □

In order to complete the proof of Theorem 3.1.2, we need to exhibit a valence grammar over M that generates a non-context-free language when given a finitely generated monoid M with infinite $R_1(M)$.

Lemma 3.4.5. *Let $R_1(M)$ be infinite for some finitely generated monoid M. Then, there is a valence grammar over M that generates a language that is not context-free.*

Proof. Let M be generated by $a_1, \ldots, a_n$ and let $X = \{x_1, \ldots, x_n\}$ be an alphabet. Furthermore, let $\varphi \colon X^* \to M$ be the surjective morphism defined by $\varphi(x_i) = a_i$. The valence grammar $G = (N, T, M, P, S_0)$ is defined as follows. Let $N = \{S_0, S_1\}$, $T = X \cup \{c\}$, and let P consist of the productions

$$(S_0 \to x_i S_0 x_i;\ a_i) \qquad (S_0 \to c S_1 c;\ 1) \qquad (S_1 \to x_i S_1;\ a_i) \qquad (S_1 \to \varepsilon;\ 1)$$

for $1 \leqslant i \leqslant n$. Then, clearly $L(G) = K = \{rcscr^R \mid r, s \in X^*,\ \varphi(rs) = 1\}$. It remains to be shown that K is not context-free. Suppose K is context-free and let m be the constant provided by Ogden's Lemma (Theorem 2.1.1). By Corollary 3.2.3, we can find an infinite subset $S \subseteq L_1 M$ such that $\overleftarrow{1}(a) \cap \overleftarrow{1}(b) = \emptyset$ for $a, b \in S$, $a \neq b$. Since φ is surjective, we can define $\ell(a)$ for every $a \in S$ to be the minimal length of a word $w \in X^*$ such that $\varphi(w)a = 1$. If $\ell(a) < m$ for all $a \in S$, the finite set $\{\varphi(w) \mid w \in X^*, |w| < m\}$ contains a left inverse for every $a \in S$. This, however, contradicts the fact that the infinitely many elements of S have disjoint sets of left inverses. Thus, there exists an $a \in S$ with $\ell(a) \geqslant m$. We choose words $r, s \in X^*$ such that $\varphi(s) = a$ and r is of minimal length among those words satisfying $\varphi(rs) = 1$. Then, by the choice of a, we have $|r| \geqslant m$.

We apply Ogden's Lemma to the word $z = rcscr^R \in K$, where we choose the first $|r|$ symbols to be marked. Let $z = uvwxy$ be the decomposition from the lemma. Condition 1 implies $|uv| < |r|$. Because of condition 4, x cannot contain a c. Furthermore, x cannot be a subword of r, since then pumping would lead to words with mismatching first and third segments. In particular, from condition 2, the first part holds and v is not empty. Thus, if x were a subword of s, pumping would again lead to a mismatching first and third segment. Hence, x is a subword of r^R. If we now pump with $i = 0$, we obtain a word $r'cscr'' \in K$, where $|r'| < |r|$. In particular, we have $\varphi(r's) = 1$, in contradiction to the choice of r. □

Theorem 3.1.2 now follows from the foregoing lemmas.

Proof of Theorem 3.1.2. Condition 2 is equivalent to condition 5 by Lemmas 3.4.4 and 3.4.5. Lemmas 3.3.3 and 3.3.4 prove that conditions 1, 3, and 4 are each equivalent to condition 5. □

3.5 Conclusion

Theorem 3.1.2 settles the question of which monoids increase the expressiveness when used as a storage mechanism in automata and grammars, provided that expressiveness is measured by the produced languages. Furthermore, for a strong notion of determinism, it describes those monoids over which valence automata can be determinized.

The main results of this chapter have appeared in [**Zetzsche2011b**].

Open problems

1. We have compared the possible behaviors of different storage mechanisms in terms of their language classes. However, language equivalence is a rather coarse notion of behavioral equivalence and there exist much more fine-grained such notions, such as bisimulation [**Milner1989**]. Hence, it would be interesting to understand which monoids increase the set of possible behaviors with respect to these *finer notions*. For example, which monoids can yield automata that are not bisimilar to finite systems?

2. We have observed that the notion of determinism for valence automata is rather strong and therefore sometimes fails to specialize to the usual notion for concrete storage mechanisms. It would therefore be interesting to study

the question of which monoids cause valence automata to have *weakly deterministic* equivalents. In order to permit actions that depend on the current monoid element, one could allow, for each configuration, at most one edge that leads to a right-invertible monoid element.

Related work The equivalence between condition 1 and condition 5 in Theorem 3.1.2 has been obtained independently by **Render2010** [**Render2010**]. In the case of groups, it was shown by **MitranaStiebe2001** [**MitranaStiebe2001**] (and essentially the same fact was observed by **Anisimov1971** [**Anisimov1971**]; see Theorem 3.3.1).

As the author learned after the submission of [**Zetzsche2011b**], the dichotomy of monoids in Proposition 3.2.2 and Corollary 3.2.3 had been well known to semigroup theorists (see, for example, [**Render2010**]). While Corollary 3.2.3 has appeared in [**Zetzsche2011b**] with the author's own proof, it is deduced here from the stronger Proposition 3.2.2, whose proof uses an idea from [**Grillet1995**].

Valence grammars (although only for certain concrete monoids) were introduced by **Paun1980** [**Paun1980**]. A thorough treatment, including normal form results and a classification of the resulting language classes for commutative monoids, has been carried out by **FernauStiebe2002a** [**FernauStiebe2002a**]. For an overview of results on valence grammars, we refer the reader to the references therein.

Acknowledgements I would like to thank Reiner Hüchting and Klaus Madlener for discussions and helpful comments which have improved the presentation of [**Zetzsche2011b**]. Furthermore, I am grateful to the anonymous referee for ICALP 2011 who made me aware that the equivalence of the second and last condition of Theorem 3.1.2 had been obtained independently by Elaine Render and that the dichotomy of monoids is well-known.

Chapter 4

Decidability of the emptiness problem

4.1 Introduction

In this chapter, we turn from questions of expressivity to those of analysis of valence automata. One of the most important problems in the algorithmic analysis of system models is the *emptiness problem*, which asks whether a given language is empty: On the one hand, this is a very natural problem. On the other hand, in most automata models, it is equivalent to the reachability problem (Given two configurations, can the second be reached from the first one?) and in the case of effective full trios, such as $\mathsf{VA}(M)$, it is equivalent to the membership problem (Given a language L and a word w, does L contain w?).

Deciding the emptiness problem is also instrumental for verifying *safety properties*: These designate certain events as undesirable and stipulate that they never occur ("*something bad never happens*") [**Berard2010**]. Hence, when an undesirable event consists in reaching a certain configuration, verifying the corresponding safety property boils down to an invocation of the reachability problem. Furthermore, if it is considered undesirable for an automaton A to perform a sequence of actions from a regular language $R \subseteq X^*$, we can verify that $L(A) \cap RX^* = \emptyset$, which is possible when the automaton model exhibits effective closure under intersection with regular sets.

We present several results concerning the decidability of the emptiness problem. In Section 4.2, we mention a connection between the emptiness problem of valence automata and the membership problem for rational subsets of monoids. Afterwards, in Section 4.3, we consider the decidability of the emptiness problem for graph monoids.

The results of this chapter have appeared in [**Zetzsche2015c**].

4.2 Groups

In this section, we mention a connection between the emptiness problem for valence automata and the rational subset membership problem for groups, which has attracted attention in recent years. This connection allows the transfer of de-

cidability results between the emptiness problem for valence automata on the one side and the membership problem for rational sets in groups on the other.

The rational subset membership problem Suppose M is a finitely generated monoid and $\varphi\colon X^* \to M$ is a surjective morphism. Then the *rational subset membership problem*[1] for M is the following decision problem: The given input is a rational subset $R \subseteq X^*$ and a word $w \in X^*$ and the question is whether $\varphi(w)$ contained in $\varphi(R)$. Note that the decidability of the problem does not depend on the chosen morphism φ.

The rational subset membership problem was subject to increased interest in recent years; see **[Lohrey2015a]** for a survey. **KambitesSilvaSteinberg2007** characterized the decidability of the emptiness problem of valence automata over a group G in terms of the group's rational subset membership problem.

Theorem 4.2.1 (KambitesSilvaSteinberg2007 [KambitesSilvaSteinberg2007]). *For valence automata over a group G, emptiness is decidable if and only if the rational subset membership problem for G is decidable.*

4.3 Graph monoids

In this section, we investigate for which graph monoids $\mathbb{M}\Gamma$, the emptiness problem is decidable for valence automata over $\mathbb{M}\Gamma$.

As a first step, we exhibit graphs Γ for which $\mathsf{VA}(\mathbb{M}\Gamma)$ includes the recursively enumerable languages.

Theorem 4.3.1. *Let Γ be a graph such that Γ^- contains C_4 or P_4 as an induced subgraph. Then $\mathsf{VA}(\mathbb{M}\Gamma)$ is the class of recursively enumerable languages. In particular, the emptiness problem is undecidable for valence automata over $\mathbb{M}\Gamma$.*

Unfortunately, it is not clear whether this describes all Γ for which $\mathsf{VA}(\mathbb{M}\Gamma)$ exhausts the recursively enumerable languages. For example, if Γ contains no loops and is just one edge short of being a clique, the monoid $\mathbb{M}\Gamma$ represents a pushdown Petri net storage. For these, it is an open problem whether the reachability problem is decidable **[Reinhardt2008]**. In Theorem 4.3.8, however, we will see that if we forbid induced subgraphs corresponding to these devices, the converse of Theorem 4.3.1 becomes true.

A result similar to Theorem 4.3.1 was shown by **LohreySteinberg2008 [LohreySteinberg2008]**: They proved that if every vertex in Γ is looped and Γ contains C_4 or P_4 as an induced subgraph, then the rational subset membership problem is undecidable for $\mathbb{M}\Gamma$. Their proof adapts a construction of **AalbersbergHoogeboom1989 [AalbersbergHoogeboom1**
which shows that the disjointness problem for rational sets of traces is undecidable when the independence relation has P_4 or C_4 as an induced subgraph. An inspection of the proof presented here, together with its prerequisites (Theorems 4.3.3 and 4.3.4), reveals that the employed ideas are very similar to the combination of **LohreySteinberg2008**'s and **AalbersbergHoogeboom1989**'s proof. Here, we use the following fact. We denote the recursively enumerable languages by RE.

[1] It should be noted that the rational subset membership problem is usually defined for groups and then with respect to a group generating set (as opposed to a monoid generating set). However, it is easy to see that the decidability of the problem is not affected by this slight deviation.

Lemma 4.3.2. *Let* $X = \{a_1, \bar{a}_1, b_1, a_2, \bar{a}_2, b_2\}$ *and let* $B_2 \subseteq X^*$ *be defined as*

$$B_2 = (\{a_1^n \bar{a}_1^n \mid n \geqslant 0\} b_1)^* \sqcup (\{a_2^n \bar{a}_2^n \mid n \geqslant 0\} b_2)^*.$$

Then RE *equals* $\mathcal{T}(B_2)$*, the smallest full trio containing* B_2*.*

Lemma 4.3.2 is essentially due to **HartmanisHopcroft1970**, who stated it in slightly different terms:

Theorem 4.3.3 (HartmanisHopcroft1970 [HartmanisHopcroft1970]). *Let* $\mathcal{C}$ *be the smallest full AFL containing* $\{a^n b^n \mid n \geqslant 0\}$*. Every recursively enumerable language is the homomorphic image of the intersection of two languages in* $\mathcal{C}$*.*

According to the following auxiliary result of **GinsburgGreibach1970 [GinsburgGreibach1970]**, Lemma 4.3.2 will follow from Theorem 4.3.3.

Theorem 4.3.4 (GinsburgGreibach1970 [GinsburgGreibach1970]). *Let* $L \subseteq X^*$ *and* $c \notin X$*. The smallest full AFL containing* L *equals* $\mathcal{T}((Lc)^*)$*.*

As announced, Lemma 4.3.2 now follows.

Proof of Lemma 4.3.2. Since clearly $\mathcal{T}(B_2) \subseteq \mathsf{RE}$, it suffices to show $\mathsf{RE} \subseteq \mathcal{T}(B_2)$. According to Theorem 4.3.3, this amounts to showing that $L_1 \cap L_2 \in \mathcal{T}(B_2)$ for any L_1 and L_2 in $\mathcal{C}$, where $\mathcal{C}$ is the smallest full AFL containing the language $S = \{a^n b^n \mid n \geqslant 0\}$. Hence, let $L_1, L_2 \in \mathcal{C}$. By Theorem 4.3.4, L_1 and L_2 belong to $\mathcal{C} = \mathcal{T}((Sc)^*)$. This means we have $L_i = T_i(\{a_i^n \bar{a}_i^n \mid n \geqslant 0\} b_i)^*$ for some rational transduction T_i for $i = 1, 2$. Using a product construction, it is now easy to obtain a rational transduction T with $L_1 \cap L_2 = TB_2$. □

The proof of Theorem 4.3.1 will require one more auxiliary lemma.

Lemma 4.3.5. *Let* $\Gamma = (V, E)$ *be a graph,* $W \subseteq V$ *a subset, and* $Y \subseteq X_\Gamma$ *be defined as* $Y = \{a_w, \bar{a}_w \mid w \in W\}$*. Then* $u \equiv_\Gamma v$ *implies* $\pi_Y(u) \equiv_\Gamma \pi_Y(v)$ *for* $u, v \in X_\Gamma^*$*.*

Proof. An inspection of the rules in the presentation T_Γ reveals that if $(u, v) \in R_\Gamma$, then either $(\pi_Y(u), \pi_Y(v)) = (u, v)$ or $\pi_Y(u) = \pi_Y(v)$. In any case, we have the equivalence $\pi_Y(u) \equiv_\Gamma \pi_Y(v)$. Since $\equiv_\Gamma$ is a congruence and π_Y a morphism, this implies the lemma. □

Note that the foregoing lemma does not hold for arbitrary alphabets $Y \subseteq X_\Gamma$. For example, if $V = \{1\}$, $X_\Gamma = \{a_1, \bar{a}_1\}$, and $Y = \{a_1\}$, then $a_1 \bar{a}_1 \equiv_\Gamma \varepsilon$, but $a_1 \not\equiv_\Gamma \varepsilon$.

We are now ready to prove Theorem 4.3.1.

Proof of Theorem 4.3.1. The definition of $\mathbb{M}\Gamma$ implies that the set of all $w \in X_\Gamma^*$ with $w \equiv_\Gamma \varepsilon$ is recursively enumerable. In particular, one can recursively enumerate runs of valence automata over $\mathsf{VA}(\mathbb{M}\Gamma)$ and hence $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{RE}$. For the other inclusion, recall that $\mathsf{VA}(M)$ is a full trio for any monoid M. Furthermore, if Δ is an induced subgraph of Γ, then $\mathbb{M}\Delta$ embeds into $\mathbb{M}\Gamma$, meaning $\mathsf{VA}(\mathbb{M}\Delta) \subseteq \mathsf{VA}(\mathbb{M}\Gamma)$. Hence, according to Lemma 4.3.2, it suffices to show that $B_2 \in \mathsf{VA}(\mathbb{M}\Gamma)$ if Γ^- equals C_4 or P_4.

Let $X = \{a_1, \bar{a}_1, b_1, a_2, \bar{a}_2, b_2\}$ and $\Gamma = (V, E)$. If Γ^- equals C_4 or P_4, then we have $V = \{1, 2, 3, 4\}$ with $\{3, 1\}, \{1, 2\}, \{2, 4\} \in E$ and $\{1, 4\}, \{2, 3\} \notin E$. See

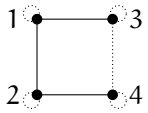


Figure 4.1: Graphs Γ where Γ^{-} is C_4 or P_4. Dotted lines represent edges that may or may not exist in Γ.

Fig. 4.1. We construct a valence automaton A over $\mathbb{M}\Gamma$ for $B_2 \subseteq X^*$ as follows. First, A nondeterministically reads a word from the regular language $R = ((a_1^* \bar{a}_1^*)b_1)^* \sqcup ((a_2^* \bar{a}_2^*)b_2)^*$. Here, when reading a_i or $\bar{a}_i$, it multiplies $[a_i]$ or $[\bar{a}_i]$, respectively, to the storage monoid. When reading b_1 or b_2, it multiplies $[a_4]$ or $[a_3]$, respectively. After this, A switches to another state and nondeterministically multiplies an element from $\{[\bar{a}_4], [\bar{a}_3]\}^*$. Then it changes into an accepting state. We shall prove that A accepts B_2. Let $h\colon X^* \to \{a_i, \bar{a}_i \mid 1 \leqslant i \leqslant 4\}^*$ be the morphism with $h(a_i) = a_i$ and $h(\bar{a}_i) = \bar{a}_i$ for $i = 1, 2$ and $h(b_1) = a_4$ and $h(b_2) = a_3$.

Suppose $w \in L(A)$. Then $w \in R$ and there is a $v \in \{\bar{a}_4, \bar{a}_3\}^*$ that satisfies $[h(w)v]_\Gamma = [\varepsilon]_\Gamma$. Let $w_i = \pi_{\{a_i, \bar{a}_i, b_i\}}(w)$. Observe that if we can show $w_i \in (\{a_i^n \bar{a}_i^n \mid n \geqslant 0\}^* b_i)^*$ for $i = 1, 2$, then clearly $w \in B_2$. For symmetry reasons, it suffices to prove this for $i = 1$. Let $Y = \{a_1, \bar{a}_1, a_4, \bar{a}_4\}$. Since $[h(w)v]_\Gamma = [\varepsilon]_\Gamma$, we have in particular $[\pi_Y(h(w)v)]_\Gamma = [\varepsilon]_\Gamma$ by Lemma 4.3.5. Moreover,

$$\pi_Y(h(w)v) = a_1^{n_1} \bar{a}_1^{\bar{n}_1} a_4 \cdots a_1^{n_k} \bar{a}_1^{\bar{n}_k} a_4 \bar{a}_4^m$$

for some $n_1, \ldots, n_k, \bar{n}_1, \ldots, \bar{n}_k, m \in \mathbb{N}$. Again, by projecting to $\{a_4, \bar{a}_4\}^*$, we obtain $[a_4^k \bar{a}_4^m]_\Gamma = [\varepsilon]_\Gamma$ and hence $k = m$. If $n_k \neq \bar{n}_k$, then it is easy to see that $\pi_Y(h(w)v)$ cannot be reduced to ε, since there is no edge $\{1, 4\}$ in Γ. Therefore, we have $n_k = \bar{n}_k$. It follows inductively that $n_i = \bar{n}_i$ for all $1 \leqslant i \leqslant k$. Since $w_i = a_1^{n_1} \bar{a}_1^{\bar{n}_1} b_1 \cdots a_1^{n_k} \bar{a}_1^{\bar{n}_k} b_1$, this implies $w_i \in (\{a_1^n \bar{a}_1^n \mid n \geqslant 0\} b_1)^*$.

We shall now prove $B_2 \subseteq L(A)$. Let $g\colon X^* \to \{\bar{a}_3, \bar{a}_4\}$ be the morphism defined by $g(a_i) = g(\bar{a}_i) = \varepsilon$ and $g(b_1) = \bar{a}_4$ and $g(b_2) = \bar{a}_3$. We show by induction on $|w|$ that $w \in B_2$ implies $[h(w)g(w)^R]_\Gamma = [\varepsilon]_\Gamma$. Since for each $w \in B_2$, A clearly has a run that puts $[h(w)g(w)^R]_\Gamma$ into the storage, this establishes $B_2 \subseteq L(A)$. Suppose $\pi_{\{b_1, b_2\}}(w)$ ends in b_1. Then $w = rsb_1$ for $r \in X^*$, $s \in (a_1^n \bar{a}_1^n) \sqcup t$ with $r \in X^*$, $n \in \mathbb{N}$ and $t \in \{a_2, \bar{a}_2, b_2\}^*$. Note that then $rt \in B_2$. Since there are edges $\{1, 2\}, \{2, 4\}$ in Γ, we have $[h(s)]_\Gamma = [h(ta_1^n \bar{a}_1^n)]_\Gamma$. Moreover, since g deletes a_1 and $\bar{a}_1$, we have $g(s) = g(t)$. Therefore,

$$\begin{aligned}[h(w)g(w)^R]_\Gamma &= [h(rsb_1)g(rsb_1)^R]_\Gamma = [h(rta_1^n \bar{a}_1^n b_1)g(rtb_1)^R]_\Gamma \\ &= [h(rt)a_1^n \bar{a}_1^n a_4 \bar{a}_4 g(rt)^R]_\Gamma = [h(rt)g(rt)^R]_\Gamma.\end{aligned}$$

By induction, we have $[h(rt)g(rt)^R]_\Gamma = [\varepsilon]_\Gamma$ and hence $[h(w)g(w)^R]_\Gamma = [\varepsilon]_\Gamma$. If $\pi_{\{b_1, b_2\}}(w)$ ends in b_2, then one can show $[h(w)g(w)^R]_\Gamma = [\varepsilon]_\Gamma$ completely analogously. This proves $B_2 \subseteq L(A)$ and hence the theorem. □

Decidability We now establish the decidability of the emptiness problem for valence automata over certain graph monoids. As already noted in Section 2.4, if $\Gamma = (V, E)$ has no loops, is one edge short of being a clique, and $|V| > 2$,

then $\mathbb{M}\Gamma \cong \mathbb{B}^{(2)} \times \mathbb{B}^{|V|-2}$. This means valence automata over $\mathbb{M}\Gamma$ are equivalent to Petri nets with $|V|-2$ unbounded places and one pushdown. In particular, the emptiness problem for valence automata is equivalent to the reachability problem for such Petri nets, for which decidability is a long-standing open problem [**Reinhardt2008**]. Therefore, characterizing those Γ with a decidable emptiness problem for valence automata over $\mathbb{M}\Gamma$ would very likely settle this open problem[2].

However, if we steer clear of pushdown Petri nets, we can achieve a characterization. More precisely, we will present a set of graphs that allow the simulation of a pushdown Petri net. Then, among those graphs that do not contain these as induced subgraphs, we characterize those for which emptiness is decidable. From Theorem 4.3.1, we already know that a P_4 or C_4 as an induced subgraph of Γ^- causes the emptiness problem to be undecidable. We will see in Theorem 4.3.8 that, if we forbid subgraphs corresponding to pushdown Petri nets, the absence of P_4 and C_4 already characterizes decidability. Let us define those subgraphs that entail the behavior of pushdown Petri nets.

PPN-graphs The graph Γ is said to be a *PPN-graph* if it is isomorphic to one of the following three graphs:

A graph Γ is called *PPN-free* is it has no PPN-graph as an induced subgraph. Observe that a graph Γ is PPN-free if and only if in the neighborhood of each unlooped vertex, any two vertices are adjacent.

Of course, the abbreviation 'PPN' refers to 'pushdown Petri nets'. This is justified by the following fact.

Proposition 4.3.6. *If Γ is a PPN-graph, then* $\mathsf{VA}(\mathbb{M}\Gamma) = \mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{B})$.

Proof. By definition, we have $\mathbb{M}\Gamma \cong \mathbb{B} \times (M_0 * M_1)$, where $M_i \cong \mathbb{B}$ or $M_i \cong \mathbb{Z}$ for $i \in \{0,1\}$. We show that $\mathsf{VA}(M_0 * M_1) = \mathsf{VA}(\mathbb{B} * \mathbb{B})$ in any case. According to Theorem 2.3.6, this implies $\mathsf{VA}(\mathbb{M}\Gamma) = \mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{B})$. If $M_0 \cong M_1 \cong \mathbb{B}$, the equality $\mathsf{VA}(M_0 * M_1) = \mathsf{VA}(\mathbb{B} * \mathbb{B})$ is trivial, so we may assume $M_0 \cong \mathbb{Z}$.

If $M_1 \cong \mathbb{Z}$, then $M_0 * M_1 \cong \mathbb{Z} * \mathbb{Z}$, meaning that $\mathsf{VA}(M_0 * M_1)$ is the class of context-free languages (Theorem 2.4.1) and thus $\mathsf{VA}(M_0 * M_1) = \mathsf{VA}(\mathbb{B} * \mathbb{B})$.

If $M_1 \cong \mathbb{B}$, then $\mathsf{VA}(\mathbb{Z} * \mathbb{B}) = \mathsf{Alg}(\mathsf{VA}(\mathbb{Z}))$ by Theorem 2.6.6. Since $\mathsf{VA}(\mathbb{Z})$ is included in the context-free languages, we have $\mathsf{Alg}(\mathsf{VA}(\mathbb{Z})) = \mathsf{VA}(\mathbb{B} * \mathbb{B})$. □

In order to exploit the absence of P_4 and C_4 as induced subgraphs, we will employ a characterization of such graphs as transitive forests.

Transitive forests The *comparability graph* of a tree t is a simple graph with the same vertices as t, but has an edge between two vertices whenever one is a descendent of the other in t. A simple graph is a *transitive forest* if it is the disjoint union of comparability graphs of trees. For an example of a transitive forest, see Fig. 4.2.

[2] Technically, it is conceivable that there is a decision procedure for each $\mathbb{B}^{(2)} \times \mathbb{B}^n$, but no uniform one that works for all n. However, this seems unlikely.

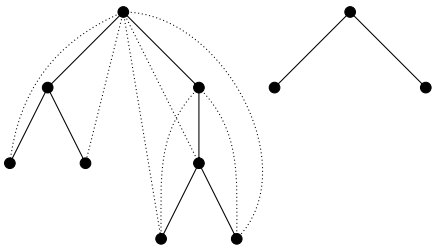


Figure 4.2: Example of a transitive forest. The solid edges are part of the trees whose comparability graphs make up the graph.

Definition 4.3.7. *By* DEC*, we denote the smallest isomorphism-closed class of monoids such that*

1. *for each* $n \geqslant 0$*, we have* $\mathbb{B}^n \in \mathsf{DEC}$ *and*
2. *for* $M, N \in \mathsf{DEC}$*, we also have* $M * N \in \mathsf{DEC}$ *and* $M \times \mathbb{Z} \in \mathsf{DEC}$*.*

Our result characterizes those PPN-free Γ for which valence automata over $\mathbb{M}\Gamma$ have decidable emptiness problem.

Theorem 4.3.8. *Let* Γ *be PPN-free. Then the following conditions are equivalent:*

1. *Emptiness is decidable for valence automata over* $\mathbb{M}\Gamma$*.*
2. Γ^- *contains neither* C_4 *nor* P_4 *as an induced subgraph.*
3. Γ^- *is a transitive forest.*
4. $\mathbb{M}\Gamma \in \mathsf{DEC}$*.*

Note that this generalizes the fact that emptiness is decidable for pushdown automata (i.e. graphs with no edges) and partially blind multicounter automata (i.e. cliques), or equivalently, reachability in Petri nets.

This theorem extends a result by **LohreySteinberg2008** [**LohreySteinberg2008**]. The latter characterizes those graph groups for which the rational subset membership problem is decidable.

Theorem 4.3.9 (**LohreySteinberg2008** [**LohreySteinberg2008**])**.** *Let* Γ *be a graph in which every vertex is looped. Then the rational subset membership problem for the group* $\mathbb{M}\Gamma$ *is decidable if and only if* Γ^- *is a transitive forest.*

According to Theorem 4.2.1, this covers the cases of Theorem 4.3.8 where in Γ every vertex is looped. **LohreySteinberg2008** show decidability by essentially proving that $\mathsf{VA}(\mathbb{M}\Gamma)$ is semilinear in their case. Here, we extend this argument by showing that in the equivalent cases of Theorem 4.3.8, the Parikh images of $\mathsf{VA}(\mathbb{M}\Gamma)$ are those of languages accepted by priority multicounter automata. The latter were introduced and shown to have a decidable reachability problem by **Reinhardt2008** [**Reinhardt2008**].

Intuition for decidable cases In order to provide an intuition for those storage mechanisms (not containing a pushdown Petri net) with a decidable emptiness problem, we present an equally expressive class of monoids for which the corresponding storage mechanisms are easier to grasp. Let $\mathsf{SC}^{\pm}$ be the smallest isomorphism-closed class of monoids with

1. for each $n \in \mathbb{N}$, we have $\mathbb{B}^n \in \mathsf{SC}^{\pm}$,
2. for each $M \in \mathsf{SC}^{\pm}$, we also have $\mathbb{B} * M \in \mathsf{SC}^{\pm}$ and $M \times \mathbb{Z} \in \mathsf{SC}^{\pm}$.

Hence, $\mathsf{SC}^{\pm}$ realizes those storage mechanisms that can be constructed from a finite set of *partially blind counters* ($\mathbb{B}^n$), *building stacks* ($M \mapsto \mathbb{B} * M$) and *adding blind counters* ($M \mapsto M \times \mathbb{Z}$). Then, in fact, the monoids in $\mathsf{SC}^{\pm}$ produce the same languages as those in DEC.

Proposition 4.3.10. $\mathsf{VA}(\mathsf{DEC}) = \mathsf{VA}(\mathsf{SC}^{\pm})$.

Proof. Since $\mathsf{SC}^{\pm} \subseteq \mathsf{DEC}$, the inclusion "$\supseteq$" is immediate. We show by induction with respect to the definition of DEC that for each $M \in \mathsf{DEC}$, there is an $M' \in \mathsf{SC}^{\pm}$ with $\mathsf{VA}(M) \subseteq \mathsf{VA}(M')$. This is trivial if $M = \mathbb{B}^n$, so suppose $\mathsf{VA}(M) \subseteq \mathsf{VA}(M')$ and $\mathsf{VA}(N) \subseteq \mathsf{VA}(N')$ for $M, N \in \mathsf{DEC}$ and $M', N' \in \mathsf{SC}^{\pm}$. Observe that by induction on the definition of $\mathsf{SC}^{\pm}$, one can show that there is a common $P \in \mathsf{SC}^{\pm}$ with $\mathsf{VA}(M') \subseteq \mathsf{VA}(P)$ and $\mathsf{VA}(N') \subseteq \mathsf{VA}(P)$. Of course, we may assume that $\mathsf{R}_1(P) \neq \{1\}$. Then we have

$$\begin{aligned}
\mathsf{VA}(M * N) &\subseteq \mathsf{Alg}(\mathsf{VA}(M) \cup \mathsf{VA}(N)) && \text{by Theorem 2.6.3} \\
&\subseteq \mathsf{Alg}(\mathsf{VA}(M') \cup \mathsf{VA}(N')) \\
&\subseteq \mathsf{Alg}(\mathsf{VA}(P)) \\
&= \mathsf{VA}(\mathbb{B} * P) && \text{by Theorem 2.6.6}
\end{aligned}$$

and $\mathbb{B} * P \in \mathsf{SC}^{\pm}$. Moreover, Corollary 2.3.7 implies $\mathsf{VA}(M \times \mathbb{Z}) \subseteq \mathsf{VA}(M' \times \mathbb{Z})$ and we have $M' \times \mathbb{Z} \in \mathsf{SC}^{\pm}$. □

Intuition for open cases We also want to provide an intuition for the remaining storage mechanisms, i.e. those defined by monoids $\mathbb{M}\Gamma$ about which Theorem 4.3.1 and Theorem 4.3.8 make no statement. To this end, we describe a class of monoids that are expressively equivalent to these remaining cases. The remaining cases are given by those graphs Γ where Γ^{-} does not contain C_4 or P_4, but Γ contains a PPN-graph. Let REM denote the class of monoids $\mathbb{M}\Gamma$, where Γ is such a graph. Let SC^{+} be the smallest isomorphism-closed class of monoids with

1. for each $n \in \mathbb{N}$, $n \geqslant 1$, we have $(\mathbb{B} * \mathbb{B}^n) \times \mathbb{B} \in \mathsf{SC}^{+}$ and
2. for each $M \in \mathsf{SC}^{+}$, we also have $\mathbb{B} * M \in \mathsf{SC}^{+}$ and $M \times \mathbb{B} \in \mathsf{SC}^{+}$.

Thus, SC^{+} realizes those storage mechanisms that are obtained from *a stack of partially blind counters, together with one partially blind counter* ($(\mathbb{B} * \mathbb{B}^n) \times \mathbb{B}$) by *building stacks* ($M \mapsto \mathbb{B} * M$) and *adding partially blind counters* ($M \mapsto M \times \mathbb{B}$). Of course, SC^{+} generalizes pushdown Petri nets, which correspond to monoids $(\mathbb{B} * \mathbb{B}) \times \mathbb{B}^n$ for $n \in \mathbb{N}$.

Proposition 4.3.11. $\mathsf{VA}(\mathsf{REM}) = \mathsf{VA}(\mathsf{SC}^{+})$.

Proving Proposition 4.3.11 requires some ingredients of the proof of Theorem 4.3.8. We therefore postpone the proof until after Theorem 4.3.8 is shown.

The remainder of this section is devoted to the proof of Theorem 4.3.8 and Proposition 4.3.11. Note that the implication "1 $\Rightarrow$ 2" immediately follows from Theorem 4.3.1. The implication "2 $\Rightarrow$ 3" is an old graph-theoretic result of Wolk.

Theorem 4.3.12 (Wolk1965 [Wolk1965]). *A simple graph Γ is a transitive forest if and only if Γ does not contain C_4 or P_4 as an induced subgraph.*

The implication "3 $\Rightarrow$ 4" is a simple combinatorial observation. An analogous fact is part of Lohrey and Steinberg's proof of Theorem 4.3.9.

Lemma 4.3.13. *If Γ is PPN-free and Γ^- is a transitive forest, then $\mathbb{M}\Gamma \in \mathsf{DEC}$.*

Proof. Let $\Gamma = (V, E)$. We proceed by induction on $|V|$. Observe that by Theorem 4.3.12, every induced subgraph of a transitive forest is again a transitive forest. Since furthermore every induced proper subgraph Δ of Γ is again PPN-free, our induction hypothesis implies $\mathbb{M}\Delta \in \mathsf{DEC}$ for such graphs. If Γ is empty, then $\mathbb{M}\Gamma \cong \mathbf{1} \cong \mathbb{B}^0 \in \mathsf{DEC}$. Hence, we assume that Γ is non-empty. If Γ is not connected, then Γ is the disjoint union of graphs Γ_1, Γ_2, for which $\mathbb{M}\Gamma_1, \mathbb{M}\Gamma_2 \in \mathsf{DEC}$ by induction. Hence, $\mathbb{M}\Gamma \cong \mathbb{M}\Gamma_1 * \mathbb{M}\Gamma_2 \in \mathsf{DEC}$. We therefore assume that Γ is connected.

Since Γ^- is a transitive forest, there is a vertex $v \in V$ that is adjacent to every vertex in $V \setminus \{v\}$. We distinguish two cases.

- If v is a looped vertex, then $\mathbb{M}\Gamma \cong \mathbb{Z} \times \mathbb{M}(\Gamma \setminus v)$, and $\mathbb{M}(\Gamma \setminus v) \in \mathsf{DEC}$ by induction.
- If v is an unlooped vertex, then Γ being PPN-free means that in $(\Gamma \setminus v)^-$, any two distinct vertices are adjacent. Hence, $\mathbb{M}\Gamma \cong \mathbb{B}^m \times \mathbb{Z}^n$ for some $m, n \in \mathbb{N}$ and thus $\mathbb{M}\Gamma \in \mathsf{DEC}$.

□

In light of Theorem 4.3.1, Theorem 4.3.12, and Lemma 4.3.13, it remains to be shown that emptiness is decidable for valence automata over monoids in DEC. We prove this by reducing the problem to the reachability problem of priority multicounter machines, whose decidability has been established by **Reinhardt2008 [Reinhardt2008]**. Priority multicounter machines are an extension of Petri nets with one inhibitor arc. Intuitively, a priority multicounter machine is a partially blind multicounter machine with the additional capability of restricted zero tests: The counters are numbered from 1 to k and for each $\ell \in \{1, \ldots, k\}$, there is a zero test instruction that checks whether counters 1 through ℓ are zero. Let us define priority multicounter machines formally.

Definition 4.3.14. *A* priority k-counter machine *is a tuple $A = (Q, X, E, q_0, F)$, where*

- Q *is a finite set of states,*
- X *is an alphabet,*
- E *is a finite subset of $Q \times X^* \times \{0, \ldots, k\} \times \mathbb{Z}^k \times Q$, and its elements are called* edges *or* transitions,

- $q_0 \in Q$ *is the* initial state, *and*
- $F \subseteq Q$ *is the set of* final states.

Elements of $Q \times X^* \times \mathbb{N}^k$ *are called* configurations. *For configurations* (q, u, μ) *and* (q', u', μ') *with* $q, q' \in Q$ *and* $\mu, \mu' \in \mathbb{N}^k$, *with* $\mu = (m_1, \ldots, m_k)$, *we write*

$$(q, u, \mu) \to_A (q', u', \mu') \quad \text{if for some } (q, x, \ell, \nu, q') \in E, \; u' = ux, \; \mu' = \mu + \nu, \text{ and } m_i = 0 \text{ for } 1 \leqslant i \leqslant \ell.$$

The language accepted by A *is defined as*

$$L(A) = \{w \in X^* \mid (q_0, \varepsilon, 0) \to_A^* (q, w, 0) \text{ for some } q \in F\}.$$

A priority multicounter machine *is a priority* k*-counter machine for some* $k \in \mathbb{N}$. *The class of languages accepted by priority multicounter machines is denoted by* Prio.

Reinhardt2008 has shown that the reachability problem for priority multicounter machines is decidable **[Reinhardt2008]**, which can be reformulated as follows.

Theorem 4.3.15 (Reinhardt2008 [Reinhardt2008]). *Emptiness is decidable for priority multicounter machines.*

The idea of the proof of "4 ⇒ 1" is, given a valence automaton over some $M \in \mathsf{DEC}$, to construct a Parikh-equivalent priority multicounter machine. This construction makes use of the following simple fact.

Lemma 4.3.16. Prio *is a Presburger closed full semi-AFL and closed under substitutions.*

Proof. The fact that Prio is a full semi-AFL can be shown by standard automata constructions. Given a priority multicounter machine A and a semilinear set $S \subseteq X^{\oplus}$, we add $|X|$ counters to A that ensure that the input is contained in $L(A) \cap \Psi^{-1}(S)$. This proves that Prio is Presburger closed.

Suppose $\sigma\colon X^* \to \mathcal{P}(Y^*)$ is a Prio-substitution. Furthermore, let A be a priority k-counter machine and let $\sigma(x)$ be given by a priority ℓ-counter machine for each $x \in X$. We construct a priority $(\ell + k)$-counter machine B from A by adding ℓ counters. B simulates A on counters $\ell + 1, \ldots, \ell + k$. Whenever A reads x, B uses the first ℓ counters to simulate the priority ℓ-counter machine for $\sigma(x)$. Using the zero test on the first ℓ counters, it makes sure that the machine for $\sigma(x)$ indeed ends up in a final configuration. Then clearly $L(B) = \sigma(L(A))$. □

In order to show that every $L \in \mathsf{VA}(M)$ for $M \in \mathsf{DEC}$ has a Parikh equivalent in Prio, we use Proposition 2.5.3 and Theorem 2.6.3. By induction with respect to the definition of DEC, it suffices to prove that

$$\Psi(\mathsf{VA}(M)), \Psi(\mathsf{VA}(N)) \subseteq \Psi(\mathsf{Prio}) \quad \text{implies} \quad \Psi(\mathsf{VA}(M \times \mathbb{Z})) \subseteq \Psi(\mathsf{Prio}) \text{ and } \Psi(\mathsf{VA}(M * N)) \subseteq \Psi(\mathsf{Prio}).$$

According to Proposition 2.5.3 and Theorem 2.6.3, this boils down to showing that $\Psi(\mathsf{SLI}(\mathsf{Prio})) \subseteq \Psi(\mathsf{Prio})$ and $\Psi(\mathsf{Alg}(\mathsf{Prio})) \subseteq \Psi(\mathsf{Prio})$. The former is a consequence of Lemma 4.3.16 and the latter is a special case of Theorem 2.6.8.

Lemma 4.3.17. *We have the effective inclusion* $\Psi(\mathsf{VA}(\mathsf{DEC})) \subseteq \Psi(\mathsf{Prio})$. *More precisely, given* $M \in \mathsf{DEC}$ *and* $L \in \mathsf{VA}(M)$, *one can construct an* $L' \in \mathsf{Prio}$ *with* $\Psi(L') = \Psi(L)$.

Proof. We proceed by induction with respect to the definition of DEC. If $M = \mathbb{B}^n$, then $\mathsf{VA}(M) \subseteq \mathsf{Prio}$, because priority multicounter machines generalize partially blind multicounter machines.

Suppose $M = N \times \mathbb{Z}$ and $\Psi(\mathsf{VA}(N)) \subseteq \Psi(\mathsf{Prio})$ and let $L \in \mathsf{VA}(M)$. By Proposition 2.5.3, we have $L = h(K \cap \Psi^{-1}(S))$ for some semilinear set S, a morphism h, and $K \in \mathsf{VA}(N)$. Hence, there is a $\bar{K} \in \mathsf{Prio}$ with $\Psi(\bar{K}) = \Psi(K)$. With this, we have $\Psi(L) = \Psi(h(\bar{K} \cap \Psi^{-1}(S)))$ and since Prio is Presburger closed, we have $h(\bar{K} \cap \Psi^{-1}(S)) \in \mathsf{Prio}$ and thus $\Psi(L) \in \Psi(\mathsf{Prio})$.

Now assume $M = M_0 * M_1$ and $\Psi(\mathsf{VA}(M_i)) \subseteq \Psi(\mathsf{Prio})$ for $i = 0, 1$ and let $L \in \mathsf{VA}(M)$. According to Theorem 2.6.3, we have $L \in \mathsf{Alg}(\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1))$. Since $\Psi(\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1)) \subseteq \Psi(\mathsf{Prio})$, Lemma 2.6.9 allows us to construct a Prio-grammar G with $\Psi(L(G)) = \Psi(L)$. By Theorem 2.6.8 and Lemma 4.3.16, this implies $\Psi(L) \in \Psi(\mathsf{Prio})$. □

The following lemma is a direct consequence of Lemma 4.3.17 and Theorem 4.3.15: Given a valence automaton over M with $M \in \mathsf{DEC}$, we construct a priority multicounter machine accepting a Parikh-equivalent language. The latter can then be checked for emptiness.

Lemma 4.3.18. *For each* $M \in \mathsf{DEC}$, *the emptiness problem for valence automata over* M *is decidable.*

Theorem 4.3.8 now follows easily.

Proof of Theorem 4.3.8. The foregoing lemmas establish the required implications as follows:

$1 \Longrightarrow 2$	by Theorem 4.3.1
$\Longrightarrow 3$	by Theorem 4.3.12
$\Longrightarrow 4$	by Lemma 4.3.13
$\Longrightarrow 1$	by Lemma 4.3.18

□

Let us now prove Proposition 4.3.11.

Proof of Proposition 4.3.11. By induction, it is easy to see that each $M \in \mathsf{SC}^+$ is isomorphic to some $\mathbb{M}\Gamma$, where Γ contains a PPN-graph and Γ^- is a transitive forest. By Theorem 4.3.12, this means Γ^- contains neither C_4 nor P_4. This proves the inclusion "$\supseteq$".

Because of Theorem 4.3.12, for the inclusion "$\subseteq$", it suffices to show that if Γ^- is a transitive forest, then there is some $M \in \mathsf{SC}^+$ with $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{VA}(M)$. We prove this by induction on the number of vertices in $\Gamma = (V, E)$. As in the proof of Lemma 4.3.13, we may assume that for every induced proper subgraph Δ of Γ, we find an $M \in \mathsf{SC}^+$ with $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{VA}(M)$.

If Γ is empty, then $\mathbb{M}\Gamma \cong \mathbf{1}$ and $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{B})$. Hence, we may assume that Γ is non-empty.

If Γ is not connected, then $\Gamma = \Gamma_1 \uplus \Gamma_2$ with graphs Γ_1, Γ_2 such that there are $M_1, M_2 \in \mathsf{SC}^+$ with $\mathsf{VA}(\mathbb{M}\Gamma_i) \subseteq \mathsf{VA}(M_i)$ for $i = 1, 2$. By induction with respect to the definition of SC^+, one can show that there is a common $N \in \mathsf{SC}^+$ with $\mathsf{VA}(M_i) \subseteq \mathsf{VA}(N)$ for $i = 1, 2$. Since then $N \neq \{1\}$, we have

$$\begin{aligned}
\mathsf{VA}(\mathbb{M}\Gamma) &= \mathsf{VA}(\mathbb{M}\Gamma_1 * \mathbb{M}\Gamma_2) \\
&\subseteq \mathsf{Alg}(\mathsf{VA}(\mathbb{M}\Gamma_1) \cup \mathsf{VA}(\mathbb{M}\Gamma_2)) && \text{by Theorem 2.6.3} \\
&\subseteq \mathsf{Alg}(\mathsf{VA}(M_1) \cup \mathsf{VA}(M_2)) \\
&\subseteq \mathsf{Alg}(\mathsf{VA}(N)) \\
&= \mathsf{VA}(\mathbb{B} * N) && \text{by Theorem 2.6.6}
\end{aligned}$$

and $\mathbb{B} * N \in \mathsf{SC}^+$.

Suppose Γ is connected. Since Γ^- is a transitive forest, there is a vertex $v \in V$ that is adjacent to every vertex in $V \setminus \{v\}$. By induction, there is an $M \in \mathsf{SC}^+$ with $\mathsf{VA}(\mathbb{M}(\Gamma \setminus v)) \subseteq \mathsf{VA}(M)$. Depending on whether v is looped or not, we have $\mathbb{M}\Gamma \cong \mathbb{M}(\Gamma \setminus v) \times \mathbb{Z}$ or $\mathbb{M}\Gamma \cong \mathbb{M}(\Gamma \setminus v) \times \mathbb{B}$. Since $\mathsf{VA}(\mathbb{Z}) \subseteq \mathsf{VA}(\mathbb{B} \times \mathbb{B})$ (one blind counter can easily be simulated by two partially blind counters), Corollary 2.3.7 yields

$$\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{VA}(\mathbb{M}(\Gamma \setminus v) \times \mathbb{B} \times \mathbb{B}) \subseteq \mathsf{VA}(M \times \mathbb{B} \times \mathbb{B})$$

and the fact that $M \times \mathbb{B} \times \mathbb{B} \in \mathsf{SC}^+$ completes the proof. □

4.4 Conclusion

In this chapter, we identified induced subgraphs C_4 and P_4 as obstructions for decidability of the emptiness problem for valence automata over graph monoids. If the monoids corresponding to pushdown Petri nets do not occur as submonoids, we have seen that C_4 and P_4 are the only obstructions.

Furthermore, for those storage mechanisms that admit a decision procedure and for those where the decidability status is open, we each provided a class of storage mechanisms that is expressively equivalent and easy to grasp. In the case where we showed decidability, these are the storage mechanisms obtained from partially blind multicounters by *building stacks* and *adding blind counters*. For those mechanisms where decidability is left open, these are the mechanisms obtained by starting from a partially blind multicounter together with a pushdown, and then applying the transformations of *building stacks* and *adding partially blind counters*.

The results in this chapter have appeared in [**Zetzsche2015c**].

Related work Since in the case of groups, the emptiness problem for valence automata is equivalent to the rational subset membership problem, the work that has been carried out on the latter is closely related to the results here. In particular, and as mentioned above, Theorem 4.3.8 extends a result on this problem for graph groups by **LohreySteinberg2008** [**LohreySteinberg2008**]. See [**Lohrey2015a**] for a survey on the rational subset membership problem.

Furthermore, **MadhusudanParlato2011** [**MadhusudanParlato2011**] propose a general model of automata with auxiliary storage. Similar in spirit to the results in Section 4.3, they exhibit a class of storage mechanisms that, when used

in automata, admit a decision procedure for the emptiness problem. This generalizes several other decidability results.

However, they also show that the realized models always satisfy a Parikh's theorem. Since this is not the case already for partially blind multicounter storages (see Section 7.2), their result very likely does not easily subsume the ones here. Note that proving that there is no encoding of our results into the other framework is impossible, since there is always a trivial encoding (use our decision procedure and output one of two fixed instances of the other framework); hence, the question is merely whether some encoding would simplify the proofs.

Open problems Section 4.3 leaves open whether the emptiness problem is decidable for the monoids from SC^+. Of course, determining the decidability status for these remaining storage mechanisms is an interesting problem. As mentioned above, this extends the open question of whether reachability is decidable for pushdown Petri nets [**Reinhardt2008**].

Observe that it is conceivable that the result of **LohreySteinberg2008** (Theorem 4.3.9), when phrased as decidability of the emptiness problem, holds in fact for arbitrary graph monoids. The latter is equivalent to the decidability for the whole class SC^+.

Chapter 5

Boolean closure

5.1 Introduction

In the previous chapters, we have observed and exploited the closure of the language classes $\mathsf{VA}(M)$ under rational transductions. In the case of regular languages, this closure property is accompanied by the closure under the Boolean operations: union, intersection, and complementation.

This combination of closure properties is useful for several reasons. First, in the case of regular languages, this particular collection is employed, for example, in the theory of automatic structures [**KhoussainovNerode1995**], since it implies that in such structures, every first-order definable relation can be represented by a regular language. Since emptiness is decidable for regular languages, one can therefore decide the first-order theory of these structures.

Second, together with the decidable emptiness problem, the effectiveness of these closure properties permit the decision of the universality problem (given a regular $R \subseteq X^*$, does R equal X^*?) and the inclusion problem (given regular R and S, does R include S?).

It is therefore an interesting question for which monoids M, the class $\mathsf{VA}(M)$ exhibits closure under the Boolean operations as this would give hope for finding new structures with a decidable first-order theory and decision procedures for valence automata. Unfortunately, this chapter answers this question in an extremely negative way. It is shown here that for finitely generated monoids M, the class $\mathsf{VA}(M)$ is not closed under the Boolean operations, unless $\mathsf{VA}(M)$ coincides with the regular languages. In fact, a much more general statement is proven: Every Boolean closed full trio that contains any non-regular language L, already includes the arithmetical hierarchy (in particular the recursively enumerable languages) relative to L. This means in a full trio beyond the regular languages, *virtually no decidability property can coexist with Boolean closure*.

Similar results as the one presented here have been obtained, for example, by Hartmanis and Hopcroft [**HartmanisHopcroft1970**]: They showed that every intersection closed full AFL that contains the language $\{a^n b^n \mid n \geqslant 0\}$ already includes the recursively enumerable languages (see Theorem 4.3.3). Moreover, Book [**Book1978**] proved that the arithmetical languages constitute the smallest Boolean closed full trio that is closed under what Book termed *homomorphic replication* (see [**Book1978**] for a definition). Hence, the result here means in Book's

result one can replace the homomorphic replication by containment of any non-regular language (which is a significantly weaker condition).

In slightly enhanced form, the results in this section have appeared in **[LohreyZetzsche2014a]**. In the latter work, it was made explicit that in order to construct every language in $\mathsf{AH}(L)$ from a given non-regular L, three fixed rational transductions suffice. Moreover, the extended version of **[LohreyZetzsche2014a]** also considers analogous questions with synchronized rational transductions instead of rational transductions **[ZetzscheLohreyKuske2015a]**.

5.2 Boolean closed full trios

In order to state the main result of this chapter, we need to define the (relative) arithmetical hierarchy. Recall that RE denotes the recursively enumerable languages. For any language class $\mathcal{C}$, we write $\mathsf{RE}(\mathcal{C})$ for the class of languages accepted by some Turing machine with an oracle $L \in \mathcal{C}$. We also write $\mathsf{RE}(L)$ for $\mathsf{RE}(\{L\})$. Then the *arithmetical hierarchy* is defined as

$$\Sigma_1 = \mathsf{RE}, \qquad \Sigma_{n+1} = \mathsf{RE}(\Sigma_n) \text{ for } n \geqslant 0, \qquad \mathsf{AH} = \bigcup_{n \geqslant 1} \Sigma_n.$$

Languages in AH are called *arithmetical*. The *arithmetical hierarchy relative to* L is defined as

$$\Sigma_1(L) = \mathsf{RE}(L), \quad \Sigma_{n+1}(L) = \mathsf{RE}(\Sigma_n(L)) \text{ for } n \geqslant 0, \quad \mathsf{AH}(L) = \bigcup_{n \geqslant 1} \Sigma_n(L).$$

For a more detailed introduction to the arithmetical hierarchy, see **[Kozen1997]**. For a language $L \subseteq X^*$, let $\alpha(L) \subseteq X$ denote the smallest subset $Y \subseteq X$ with $L \subseteq Y^*$. Then, the *complement* of L is the language $\overline{L} = \alpha(L)^* \setminus L$. A language class $\mathcal{C}$ is *Boolean closed* if for each $K, L \in \mathcal{C}$, we have $K \cup L \in \mathcal{C}$ and $\overline{L} \in \mathcal{C}$.

We are now in a position to formulate the main result of this chapter.

Theorem 5.2.1. *Let L be a non-regular language. Then $\mathsf{AH}(L)$ is the smallest Boolean closed full trio containing L.*

Before we prove Theorem 5.2.1, we record some consequences. The first one applies to a wide range of language classes. Although the author is not aware of any particular full semi-AFL for which it is not known whether complementation closure is available, the following fact is interesting because of its generality.

Corollary 5.2.2. *Other than the regular languages, no full semi-AFL $\mathcal{C} \subseteq \mathsf{RE}$ is closed under complementation.*

Proof. Suppose $\mathcal{C}$ were a complementation closed full semi-AFL that contains a non-regular language. According to Theorem 5.2.1, it would already include AH and thus not be included in RE. □

Note that the next corollary is not a special case of Corollary 5.2.2 as it is not restricted to language classes below RE.

Corollary 5.2.3. *A principal full trio is closed under complementation if and only if it coincides with the regular languages.*

Proof. Consider the principal full trio $\mathcal{T}(L)$. If L is regular, $\mathcal{T}(L)$ coincides with the regular languages and is therefore closed under complementation.

Suppose L is not regular. $\mathcal{T}(L)$ consists of all languages of the form RL, where R is a rational transduction. Hence, $\mathcal{T}(L)$ is contained in $\mathsf{RE}(L)$ and closed under union. If $\mathcal{T}(L)$ were closed under complementation, it would be closed under all Boolean operations and thus, by Theorem 5.2.1, include $\mathsf{AH}(L)$. Since $\mathsf{RE}(L) \subsetneq \mathsf{AH}(L)$, this is a contradiction. □

For each finitely generated monoid M, the class $\mathsf{VA}(M)$ is a principal full trio (Corollary 2.3.5). Therefore, together with Theorem 3.1.2, Corollary 5.2.3 implies the following.

Corollary 5.2.4. *For finitely generated monoids M, the following are equivalent:*

1. $\mathsf{VA}(M)$ *is closed under complementation.*
2. $\mathsf{VA}(M)$ *coincides with the regular languages.*
3. $\mathsf{R}_1(M)$ *is finite.*

We shall prove Theorem 5.2.1 by constructing every language in $\mathsf{AH}(L)$ from L using rational transductions and the Boolean operations. In order to construct the recursively enumerable languages, we rely on the well-known fact that each of them is accepted by some two-counter machine **[Minsky1961]**.

Two-counter machines We define the alphabet $\Delta = \{+, -, z\}$, whose elements will represent the operations *increment*, *decrement*, and *zero test*, respectively.

A *two-counter machine* is a tuple $A = (Q, X, E, q_0, F)$, where Q is a finite set of *states*, X is its *input alphabet*, $E \subseteq Q \times X^* \times \Delta \times \Delta \times Q$ is a finite set of *edges*, $q_0 \in Q$ is its *initial state*, and $F \subseteq Q$ is its set of *final states*. A *configuration* is an element of $Q \times X^* \times \mathbb{N} \times \mathbb{N}$. For configurations (q, u, n_0, n_1) and (q', u', n_0', n_1'), we write $(q, u, n_0, n_1) \to_A (q', u', n_0', n_1')$ if there is an edge $(q, v, \delta_0, \delta_1, q') \in E$ such that $u' = uv$ and for each $i \in \{0, 1\}$, we have

1. $\delta_i = +$ and $n_i' = n_i + 1$,
2. $\delta_i = -$ and $n_i' = n_i - 1$, or
3. $\delta_i = z$ and $n_i' = n_i = 0$.

The language *accepted* by A is then

$$L(A) = \{w \in X^* \mid (q_0, \varepsilon, 0, 0) \to_A^* (f, w, n_0, n_1) \text{ for some } f \in F \text{ and } n_0, n_1 \in \mathbb{N}\}.$$

The definition here forces the machine to operate on both counters in each step, whereas in the usual definition, these automata can also use only one counter at a time. This is not a serious restriction: A two-counter machine that sometimes accesses only one counter at a time can be simulated as follows. First, we change it so that it always uses only one counter at a time. Then, instead of incrementing counter i, we first increment both counters and then decrement counter $1 - i$ and increment counter i again. If we proceed analogously for decrement (decrement i and increment $1 - i$, then decrement i and decrement $1 - i$) and zero test (zero

test on i and increment on $1-i$, then zero test on i and decrement on $1-i$), we represent the counter values (n_0, n_1) of the old machine by the values $(2n_0, 2n_1)$ and thus accept the same language.

In order to construct languages accepted by two-counter automata, we construct the language $C \subseteq \Delta^*$ of words that describe valid sequences of counter operations.

Definition 5.2.5. *Let $C \subseteq \Delta^*$ be the set of words $\delta_1 \cdots \delta_m$, $\delta_1, \ldots, \delta_m \in \Delta$, for which there are numbers $x_0, \ldots, x_m \in \mathbb{N}$ such that $x_0 = 0$ and for $1 \leqslant i \leqslant m$:*

1. *if $\delta_i = +$, then $x_i = x_{i-1} + 1$,*
2. *if $\delta_i = -$, then $x_i = x_{i-1} - 1$, and*
3. *if $\delta_i = z$, then $x_i = x_{i-1} = 0$.*

The main difficulty in proving Theorem 5.2.1 is to construct C from a language L, where the only information we have about L is that it is not regular. The key idea of the construction is to use the characterization of regular languages as those that have infinitely many Myhill-Nerode classes. Let X be an alphabet and $L \subseteq X^*$. For words $u, v \in X^*$, we write $u \equiv_L v$ if for each $w \in X^*$, we have

$$uw \in L \quad \text{if and only if} \quad vw \in L.$$

The equivalence relation $\equiv_L$ is called the *Myhill-Nerode equivalence*. The well-known Myhill-Nerode Theorem [**Kozen1997**] states that L is regular if and only if $\equiv_L$ has a finite index. Using the Myhill-Nerode equivalence, we define another language, which can be thought of as encoding counter values as Myhill-Nerode classes.

Definition 5.2.6. *Suppose the alphabets X, Δ, and $\{\#\}$ are pairwise disjoint. We define $\hat{C}_L \subseteq (\Delta \cup X \cup \{\#\})^*$ to be the set of all words*

$$v_0 \delta_1 v_1 \cdots \delta_m v_m \# u_0 \# \cdots u_n \#$$

with $\delta_i \in \Delta$, $v_i \in X^*$, $u_j \in X^*$, such that $u_k \not\equiv_L u_\ell$ for $k \neq \ell$, $v_0 \equiv_L u_0$, and for each $1 \leqslant i \leqslant m$, we have

1. *if $\delta_i = +$ and $v_{i-1} \equiv_L u_j$, then $v_i \equiv_L u_{j+1}$,*
2. *if $\delta_i = -$ and $v_{i-1} \equiv_L u_j$, $v_i \equiv_L u_{j-1}$, and*
3. *if $\delta_i = z$, then $v_{i-1} \equiv_L v_i \equiv_L u_0$.*

Hence, the words $v_0, \ldots, v_m$ describe the counter values as they are attained over time (the class of v_i represents the value at time $i \in \{0, \ldots, m\}$), and the words $u_0, \ldots, u_n$ describe the counter values sorted by their magnitude (the class u_j represents the value $j \in \{0, \ldots, n\}$).

It is not hard to see that $\pi_\Delta(\hat{C}_L)$ contains valid sequences of counter operations, but only those whose counter values remain below the index of $\equiv_L$. In particular, if L is not regular, $\hat{C}_L$ describes all valid sequences of counter operations.

Lemma 5.2.7. *If L is not regular, then $\pi_\Delta(\hat{C}_L) = C$.*

Proof. In order to prove the inclusion "$\supseteq$", let $x_0, \ldots, x_m \in \mathbb{N}$ be numbers as in the definition of C and suppose $\{x_0, \ldots, x_m\} \subseteq \{0, \ldots, n\}$. Since L is not regular, we can find words $u_0, \ldots, u_n \in X^*$ such that $u_k \not\equiv_L u_\ell$ for $k \neq \ell$. Now for each $0 \leqslant i \leqslant m$, let $v_i = u_{x_i}$. Then it can be checked straightforwardly that $v_0\delta_1 v_1 \cdots \delta_m v_m \# u_0 \# \cdots u_n \# \in \hat{C}_L$ and hence $\delta_1 \cdots \delta_m \in \pi_\Delta(\hat{C}_L)$.

For the inclusion "$\subseteq$", suppose $\delta_1 \cdots \delta_m \in \pi_\Delta(\hat{C}_L)$. Then there are words $v_0, \ldots, v_m \in X^*, u_0, \ldots, u_n \in X^*$ with

$$v_0\delta_1 v_1 \cdots \delta_m v_m \# u_0 \# \cdots u_n \# \in \hat{C}_L.$$

Using the fact that the u_k are pairwise incongruent w.r.t. $\equiv_L$ and by induction on i, one can easily verify that for each $0 \leqslant i \leqslant m$, there is a unique $x_i \in \{0, \ldots, n\}$ such that $v_i \equiv_L u_{x_i}$. By the definition of $\hat{C}_L$, this choice of $x_0, \ldots, x_n$ satisfies the conditions 1 to 3 of Definition 5.2.5. □

The following lemma is the key ingredient in the proof of Theorem 5.2.1. It will be convenient to denote the smallest Boolean closed full trio containing L by $\mathcal{BT}(L)$.

Lemma 5.2.8. *Let $L \subseteq X^*$ be non-regular. Then C is in $\mathcal{BT}(L)$.*

Proof. By Lemma 5.2.7, it suffices to show that $\hat{C}_L$ is in $\mathcal{BT}(L)$. We will use the alphabet $Y = X \cup \{\#\} \cup \Delta$, where we assume that X, Δ, and $\{\#\}$ are pairwise disjoint. In the following, when we say that a language K can be *constructed*, we mean that K can be obtained from L using rational transductions and the Boolean operations.

There are clearly rational transductions T_1 and T_2 with

$$\begin{aligned} W_1 &= \{u\#v\#w \mid u, v, w \in X^*,\ uw \in L\} = T_1 L, \\ W_2 &= \{u\#v\#w \mid u, v, w \in X^*,\ vw \in L\} = T_2 L, \end{aligned}$$

which means $W_1, W_2 \in \mathcal{BT}(L)$. Hence,

$$\begin{aligned} W' &= \{u\#v\#w \mid u, v, w \in X^*, (uw \in L, vw \notin L) \text{ or } (uw \notin L, vw \in L)\} \\ &= (W_1 \cap \overline{W_2}) \cup (\overline{W_1} \cap W_2) \end{aligned}$$

is in $\mathcal{BT}(L)$ as well. We can clearly find a rational transduction T_3 with

$$\begin{aligned} W &= \{u\#v \mid u, v \in X^*, u \not\equiv_L v\} \\ &= \{u\#v \mid u\#v\#w \in W' \text{ for some } w \in X^*\} = T_3 W'. \end{aligned}$$

This means $P = \{u\#v \mid u \equiv_L v\} = X^*\#X^* \setminus W = T_4\overline{W}$, for some T_4, belongs to $\mathcal{BT}(L)$. With suitable rational transductions T_5, T_6, we have

$$\begin{aligned} S &= \{u_0\#u_1\# \cdots u_n\# \mid u_i \not\equiv_L u_j \text{ for all } i \neq j\} \\ &= (X^*\#)^* \setminus \{ru\#sv\#t \mid r, s, t \in (X^*\#)^*, u\#v \in P\} = T_6\overline{T_5 P}, \end{aligned}$$

meaning that $S \in \mathcal{BT}(L)$. Let M (*matching*) be the set of all words $v_1\delta v_2\#u_1\#u_2$ where $v_1, v_2, u_1, u_2 \in X^*$ with

1. if $\delta = +$, then $v_1 \equiv_L u_1$ and $v_2 \equiv_L u_2$,

2. if $\delta = -$, then $v_1 \equiv_L u_2$ and $v_2 \equiv_L u_1$, and
3. if $\delta = z$, then $v_1 \equiv_L v_2 \equiv_L u_1$.

Since

$$\begin{aligned} M &= \{v_1 + v_2 \# u_1 \# u_2 \mid v_1 \# u_1 \in P,\ v_2 \# u_2 \in P\} \\ &\quad \cup \{v_1 - v_2 \# u_1 \# u_2 \mid v_1 \# u_2 \in P,\ v_2 \# u_1 \in P\} \\ &\quad \cup \{v_1 z v_2 \# u_1 \# u_2 \mid v_1 \# v_2 \in P,\ v_1 \# u_1 \in P,\ u_2 \in X^*\} \\ &= (T_7 P \cap T_8 P) \cup (T_9 P \cap T_{10} P) \cup (T_{11} P \cap T_{12} P) \end{aligned}$$

for suitable rational transductions $T_7, \ldots, T_{12}$, we have $M \in \mathcal{BT}(L)$.

Let E (*error*) be the set of words $v_1 \delta v_2 \# u_0 \# \cdots u_n \#$ such that for every index $1 \leqslant j \leqslant n$, we have $v_1 \delta v_2 \# u_{j-1} \# u_j \notin M$ or we have $\delta = z$ and $v_1 \not\equiv_L u_0$. Since

$$E' = \{v_1 \delta v_2 \# r u_1 \# u_2 \# s \mid v_1 \delta v_2 \# u_1 \# u_2 \in M, r, s \in (X^* \#)^*\} = T_{13} M$$

for some rational transduction T_{13}, we find $E' \in \mathcal{BT}(L)$. Furthermore, since

$$\begin{aligned} E &= \{v_1 z v_2 \# u_0 \# r \mid v_1 \not\equiv_L u_0,\ r \in (X^* \#)^*, v_2 \in X^*\} \cup \left[(X^* \Delta X^* \# (X^* \#)^* \setminus E'\right] \\ &= T_{14} \overline{P} \cup T_{15} \overline{E'}, \end{aligned}$$

for some rational transductions T_{14}, T_{15}, we have $E \in \mathcal{BT}(L)$.

Let N (*no error*) be the set of words $v_0 \delta_1 v_1 \cdots \delta_m v_m \# u_0 \# \cdots u_n \#$ such that for every $1 \leqslant i \leqslant m$, there is a $1 \leqslant j \leqslant n$ with $v_{i-1} \delta_i v_i \# u_{j-1} \# u_j \in M$ and if $\delta_i = z$, then $v_{i-1} \equiv_L u_0$. Since

$$\begin{aligned} N' &= \{w \in (X^* \Delta)^* v_1 \delta v_2 (\Delta X^*)^* \# u_0 \# \cdots u_n \# \mid v_1 \delta v_2 \# u_0 \# \cdots u_n \# \in E\} = T_{16} E, \\ N &= (X^* \Delta)^+ X^* \# (X^* \#)^* \setminus N' = T_{17} \overline{N'} \end{aligned}$$

for some rational transductions T_{16}, T_{17}, we find $N \in \mathcal{BT}(L)$.

Finally, the language I (*initial condition*) is defined as the set of all words $v_0 \delta_1 v_1 \cdots \delta_m v_m \# u_0 \# \cdots u_n \# \in N$ such that $v_0 \equiv_L u_0$. Since

$$I = N \cap \{v_0 (\Delta X^*)^* \# u_0 \# (X^* \#)^* \mid v_0 \# u_0 \in P\} = N \cap T_{18} P,$$

for some rational transduction T_{18}, we have $I \in \mathcal{BT}(L)$.

Now we have $\hat{C}_L = I \cap (X^* \Delta)^* X^* \# S = N \cap T_{19} S$ for some rational transduction T_{19}, meaning $\hat{C}_L \in \mathcal{BT}(L)$. By Lemma 5.2.7, we have $C = T_{20} \hat{C}_L$ for some rational transduction T_{20}. This proves $C \in \mathcal{BT}(L)$. □

Now that Lemma 5.2.8 is established, the remainder of the proof of Theorem 5.2.1 requires only standard arguments.

Lemma 5.2.9. *Let* L *be non-regular. Then* $\mathsf{RE} \subseteq \mathcal{BT}(L)$.

Proof. Suppose $K \subseteq X^*$ is recursively enumerable and let $A = (Q, X, E, q_0, F)$ be a two-counter machine accepting K with $Q = \{0, \ldots, k\}$ and $F = \{k\}$. Let R be the regular language of all words

$$0^{m_0} \prod_{i=1}^{n} \# w_i \# \delta_i^{(0)} \delta_i^{(1)} 0^{m_i}$$

with $(m_{i-1}, w_i, \delta_i^{(0)}, \delta_i^{(1)}, m_i) \in E$ for every $1 \leqslant i \leqslant n$, $m_0 = 0$, and $m_n = k$. Since R is regular, we have $R \in \mathcal{BT}(L)$. Clearly, there are rational transductions T_1 and T_2 such that

$$\begin{aligned} U &= \left\{ 0^0 \prod_{i=1}^{n} \#w_i\#\delta_i^{(0)}\delta_i^{(1)}10^{m_i} \in R \,\middle|\, \delta_1^{(k)} \cdots \delta_n^{(k)} \in C \text{ for } k = 0, 1 \right\} \\ &= R \cap T_1 C \cap T_2 C, \end{aligned}$$

meaning that $U \in \mathcal{BT}(L)$. Finally, applying to U the transduction T_3 that outputs all occurrences of X after odd occurrences of # up to the next occurrence of # clearly yields K, implying $K \in \mathcal{BT}(L)$. □

We are now ready to prove Theorem 5.2.1.

Proof of Theorem 5.2.1. We shall prove that for any $K \subseteq X^*$, we have the inclusion $\mathsf{RE}(K) \subseteq \mathcal{BT}(\{K, L\})$. This clearly implies $\Sigma_1(L) = \mathsf{RE}(L) \subseteq \mathcal{BT}(L)$ and hence, by induction on i, all of $\Sigma_i(L) \subseteq \mathcal{BT}(L)$.

Let $M \in \mathsf{RE}(K)$. Without losing generality, we may assume $M \subseteq X^*$ (if this is not the case, enlarge X). This means there is an oracle Turing machine A with access to a K-oracle such that M is accepted by A. We will use the extended alphabet $Y = X \cup \{\#_1, \#_2\}$, in which $X \cap \{\#_1, \#_2\} = \emptyset$. Let $M' \subseteq Y^*$ be the set of words

$$u_1\#_1 \cdots u_n\#_1 v_1\#_2 \cdots v_m\#_2 w$$

such that there is an accepting computation in A with input $w \in X^*$ and in which oracle queries about $u_1, \ldots, u_n$ are made with a positive result and oracle queries about $v_1, \ldots, v_m$ are made with a negative result. Note that this does not mean that $u_i \in K$ or $v_i \notin K$, we collect all computations that A could make and what inputs would be accepted provided that an oracle answered as specified. Then M' is clearly recursively enumerable and contained in $\mathcal{BT}(\{L\})$ by Lemma 5.2.9.

Furthermore, since

$$(K\#_1)^* = \overline{(X^*\#_1)^*\overline{K}\#_1(X^*\#_1)} = \overline{T_1\overline{K}}, \qquad (\overline{K}\#_2)^* = \overline{(X^*\#_2)^*K\#_2(X^*\#_2)} = \overline{T_2 K}$$

for suitable rational transductions T_1, T_2, we have $(K\#_1)^*, (\overline{K}\#_2)^* \in \mathcal{BT}(\{K, L\})$. Moreover, since

$$\begin{aligned} M'' &= \{u_1\#_1 \cdots u_n\#_1 v_1\#_2 \cdots v_m\#_2 w \in M' \mid w \in X^*, \\ &\qquad\qquad u_1, \ldots, u_n \in K, \\ &\qquad\qquad v_1, \ldots, v_m \in \overline{K}\} \\ &= M' \cap (K\#_1)^*(X^*\#_2)^*X^* \cap (X^*\#_1)^*(\overline{K}\#_2)^*X^* \\ &= M' \cap T_3(K\#_1)^* \cap T_4(\overline{K}\#_2)^* \end{aligned}$$

for suitable rational transductions T_3, T_4, we have $M'' \in \mathcal{BT}(\{K, L\})$. If we now apply a transduction T_5 that for an input from Y^* outputs the longest suffix in X^*, we obtain $M \in \mathcal{BT}(\{K, L\})$, completing the proof of $\mathsf{AH}(L) \subseteq \mathcal{BT}(L)$.

The inclusion $\mathcal{BT}(L) \subseteq \mathsf{AH}(L)$ follows by observing that $\mathsf{AH}(L)$ is a Boolean closed full trio. □

5.3 Conclusion

In this chapter, we have asked which storage mechanisms cause the induced language class to be closed under Boolean operations. We have given an answer that concerns significantly more language classes than those of valence automata. Specifically, we have shown that given an arbitrary non-regular language, one can construct the whole arithmetical hierarchy using just rational transductions and Boolean operations. Since this hierarchy goes far beyond the recursively enumerable languages, this tells us that the regular languages constitute the only language class that is closed under rational transductions and Boolean operations and exhibits any form of decidability. Our proof uses an encoding of counter values as Myhill-Nerode classes.

The results of this chapter have appeared in [**LohreyZetzsche2014a**, **ZetzscheLohreyKuske2015a**].

Open problems

- One of the original motivations for studying Boolean closed full trios is the following. The theory of automatic structures [**KhoussainovNerode1995**] employs closure properties of the regular languages to show that in these structures, every first-order definable relation can be represented by a regular language. The employed closure properties are subsumed by the Boolean operations and rational transductions, so that identifying language classes with this combination of closure properties and decidable emptiness might lead to new structures with decidable first-order theory.

  While the result here proves this impossible, one can observe that not the full power of rational transductions is required to obtain all first-order definable relations. Therefore, the question arises whether a smaller set of transductions allows the construction of all first-order definable relations and admits a decision procedure.

  A natural candidate for such a smaller set of transductions is that of synchronized rational transductions. The work [**ZetzscheLohreyKuske2015a**] of the author, Markus Lohrey, and Dietrich Kuske shows that then, both situations can occur: Some non-regular languages make the emptiness problem undecidable, but not every non-regular language. However, it is not clear which non-regular languages precisely permit decidability.

- There are several ways in which one could attempt to generalize Theorem 5.2.1 further. One consequence is that for each language L, for expressions involving L, rational transductions, and Boolean operations, emptiness is undecidable. If one could show that this is also the case when using rational transductions first and then Boolean operations afterwards, a disjunctive normal form transformation would yield that also the following problem is undecidable: Given rational transductions $T_1, \ldots, T_n$ and $U_1, \ldots, U_m$, does the inclusion $T_1 L \cap \cdots \cap T_n L \subseteq U_1 L \cup \cdots \cup U_m L$ hold?

  Or, even stronger, is the inclusion problem for $\mathcal{T}(L)$ undecidable for every non-regular L? Precisely stated, the *inclusion problem for* $\mathcal{T}(L)$ is the following: Given rational transductions T and U, does $TL \subseteq UL$? We suspect that this is the case. A restriction of this question is whether the inclusion problem for every class $\mathsf{VA}(M)$ is undecidable, unless $\mathsf{VA}(M) = \mathsf{Reg}$.

In Section 11.2 we will discuss a result of **Render2010** [**Render2010**], which states that $\mathsf{VA}(M)$ always either contains $\mathsf{VA}(\mathbb{B})$ or $\mathsf{VA}(\mathbb{Z})$ or it equals $\mathsf{VA}(G)$ for some torsion group G. Since the inclusion problem is undecidable for $\mathsf{VA}(\mathbb{B})$ and $\mathsf{VA}(\mathbb{Z})$, the only remaining case is that of (infinite) torsion groups.

Related work Describing language classes by a set of contained generating languages and a collection of closure properties is one of the chief ideas of AFL theory [**Ginsburg1975**]. **Jantzen1979** [**Jantzen1979**], for example, obtained such characterizations of the Petri net languages. Descriptions of this kind for the recursively enumerable languages were presented by **HausslerZeiger1980** [**HausslerZeiger1980**] and **HartmanisHopcroft1970** [**HartmanisHopcroft1970**]. A survey of language classes that are principal trios has been prepared by **Reinhardt2015** [**Reinhardt2015**].

More specifically, there is another such characterization of the arithmetical hierarchy. **Book1978** [**Book1978**] has shown that it constitutes the smallest Boolean closed full trio that is closed under an operation he calls 'homomorphic replication'. Since this operation is easily seen to produce non-regular languages, Theorem 5.2.1 subsumes **Book1978**'s result.

Acknowledgements I am grateful to Markus Lohrey for discussions on the main result, which clarified parts of the construction and lead to other results in [**LohreyZetzsche2014a**, **ZetzscheLohreyKuske2015a**].

Chapter 6

Context-freeness

6.1 Introduction

In this chapter, we study which monoids M cause $\mathsf{VA}(M)$ to contain only context-free languages. In the case of graph monoids, the situation is simple: It is easy to see that $\mathsf{VA}(\mathbb{M}\Gamma)$ is included in CF if and only if is contains no edges aside from loops. Therefore, we consider a much larger class of monoids, namely arbitrary graph products. We present a characterization of those graph products (of monoids) whose corresponding valence automata accept only context-free languages. Graph products are a generalization of the free and the direct product in the sense that for each pair of participating factors, it can be specified whether they should commute in the product.

Valence automata over a group accept only context-free languages if and only if the group's word problem (and hence the group itself) can be described by a context-free grammar. Thus, a characterization of the desired kind had already been available for groups in a result by **LohreySenizergues2007** [**LohreySenizergues2007**] (see Theorem 6.3.2). Therefore, our characterization can be regarded as an extension of Lohrey and Sénizergues' to monoids.

The results of this chapter have appeared in [**BuckheisterZetzsche2013a**].

6.2 Graph products

Let $\Gamma = (V, E)$ be a simple graph and M_v a monoid for each $v \in V$ such that $M_u \cap M_v = \emptyset$ for $u \neq v$. Moreover, let P be the free product of all the M_v for $v \in V$. By $\approx_\Gamma$, we denote the smallest congruence on M such that

$$xy \approx_\Gamma yx \qquad \text{for all } x \in M_u,\ y \in M_v, \text{ where } \{u, v\} \in E.$$

Then the *graph product* $\mathbb{M}(\Gamma, (M_v)_{v\in V})$ is defined as

$$\mathbb{M}(\Gamma, (M_v)_{v\in V}) \;=\; P/{\approx_\Gamma}.$$

In other words, M_u and M_v commute in $\mathbb{M}(\Gamma, (M_v)_{v\in V})$ if and only if u and v are adjacent (unless, of course, M_u or M_v is trivial). Clearly, if Γ is an anti-clique, $\mathbb{M}(\Gamma, (M_v)_{v\in V})$ is the free product of the M_v. If Γ is a clique, then the graph

product is the direct product of the M_v. Moreover, if each M_v is either $\mathbb{B}$ or $\mathbb{Z}$, then $\mathbb{M}(\Gamma, (M_v)_{v\in V})$ is a graph monoid.

For the sake of simplicity, we say that a monoid M is *context-free* if $\mathsf{VA}(M)$ contains only context-free languages. In order to establish context-freeness of a graph product, we will express it as a free product with amalgamation and then apply the fact that the latter product preserves context-freeness (Theorem 2.6.3). The following lemma explains how to decompose graph products into such free products with amalgamation.

Let Γ be a simple graph. If Γ has induced subgraphs Γ_0, Γ_1, and S such that $\Gamma_0 \cap \Gamma_1 = S$ and $\Gamma = \Gamma_0 \cup \Gamma_1$, we say that Γ arises from Γ_0 and Γ_1 by *pasting* these graphs together along S **[Diestel2010]**. Suppose $M = \mathbb{M}(\Gamma, (M_v)_{v\in V})$. In slight abuse of notation, we then write $M{\restriction_\Delta}$ for the graph product $\mathbb{M}(\Delta, (M_v)_{v\in V(\Delta)})$, if Δ is a subgraph of Γ. The following lemma is a slight generalization of an insight of Green **[Green1990]**, where the decomposition was proved in the case that all factors are groups, Γ_0 consists of one vertex, and S is its neighborhood.

Lemma 6.2.1. *Let $\Gamma = (V, E)$ and $M = \mathbb{M}(\Gamma, (M_v)_{v\in V})$ be a graph product such that Γ arises by pasting Γ_0 and Γ_1 together along S. Then $M \cong M{\restriction_{\Gamma_0}} *_{M{\restriction_S}} M{\restriction_{\Gamma_1}}$.*

Proof. Suppose that for each $v \in V$, we have a presentation (A_v, R_v) for M_v such that $A_u \cap A_v = \emptyset$ for $u \neq v$. Moreover, for a subgraph Δ of Γ, let

$$C_\Delta = \{(ab, ba) \mid a \in A_u, b \in A_v, \{u, v\} \in E(\Delta)\}.$$

Then M is presented by (A, R), where

$$A = \bigcup_{v\in V} A_v, \qquad R = \bigcup_{v\in V} R_v \cup C_\Gamma.$$

For each $a \in A$, let $\bar{a}$ be a new symbol and let $\bar{X} = \{\bar{x} \mid x \in X\}$ for each subset $X \subseteq A$, $\bar{w} = \bar{w}_1 \cdots \bar{w}_n$ for each $w \in A^*$, $w = w_1 \cdots w_n$ with $w_1, \ldots, w_n \in A$ and $\bar{T} = \{(\bar{v}, \bar{w}) \mid (v, w) \in T\}$ for each relation $T \subseteq A^* \times A^*$. Then the monoid $M{\restriction_{\Gamma_0}} *_{M{\restriction_S}} M{\restriction_{\Gamma_1}}$ is presented by (A', R'), where $A' = A \cup \bar{A}$ and

$$R' = \bigcup_{v\in V(\Gamma_0)} R_v \cup \bigcup_{v\in V(\Gamma_1)} \bar{R}_v \cup C_{\Gamma_0} \cup \bar{C}_{\Gamma_1} \cup \bigcup_{v\in V(S)} \{(a, \bar{a}) \mid a \in A_v\}.$$

Let $\varphi\colon A'^* \to A^*$ be the morphism with $\varphi(a) = \varphi(\bar{a}) = a$ for $a \in A$. It is readily verified that $u \equiv_{R'} v$ if and only if $\varphi(u) \equiv_R \varphi(v)$: The "only if" is clear by definition of R and R'. For the "if" direction, note that every edge of Γ lies in Γ_0 or in Γ_1. Therefore, we can match any rule in C_Γ by applying a rule in C_{Γ_0} or C_{Γ_1} and (if necessary) some rules $(a, \bar{a})$ with $a \in A_v$, $v \in V(S)$. Moreover, the rules in R_v can be matched by those in R_v, $v \in V(\Gamma_0)$, or R_v, $v \in V(\Gamma_1)$, and (if necessary) some rules $(a, \bar{a})$, $a \in A_v$, $v \in V(S)$. Hence, the monoids presented by (A, R) and (A', R') are isomorphic, which proves $M \cong M{\restriction_{\Gamma_0}} *_{M{\restriction_S}} M{\restriction_{\Gamma_1}}$. □

6.3 Context-freeness for groups

The class of finitely generated groups for which $\mathsf{VA}(G)$ contains only context-free languages has been subject to intensive study for the following reason. Note that $\mathsf{VA}(G) \subseteq \mathsf{CF}$ is equivalent to the context-freeness of all identity languages

of G (Theorem 2.3.3). In the case of a finitely generated group, each identity language describes the group completely, since two elements g_1 and g_2 of G are equal if and only if the word representing $g_1 g_2^{-1}$ belongs to the identity language. This means, if G is a finitely generated group, then $\mathsf{VA}(G) \subseteq \mathsf{CF}$ if and only if there is a context-free grammar that describes an identity language of G and hence G itself. Therefore, such groups have been called 'context-free' independently of the study of valence automata and have received considerable attention; see [**DiekertWeiss2013**] for a survey. It should be noted that in the case of groups, an identity language with respect to a generating set is also called *word problem*.

The structure of context-free groups is very well understood, as demonstrated by the following result. A *finite index subgroup* is a subgroup H of a group G such that the equivalence relation $\sim$ with $g_1 \sim g_2$ if and only if $g_1 g_2^{-1} \in H$ has finitely many equivalence classes. A group is *free (of rank n)* if it is isomorphic to a group $\mathbb{Z}^{(n)}$. A finitely generated group is said to be *virtually free* if it has a finite index subgroup that is free. The following is a very well-known result by Muller and Schupp [**MullerSchupp1983**] and Dunwoody [**Dunwoody1985**].

Theorem 6.3.1 (Muller, Schupp [**MullerSchupp1983**], Dunwoody [**Dunwoody1985**])**.** *A finitely generated group is context-free if and only if it is virtually free.*

Therefore, our question of when $\mathsf{VA}(M) \subseteq \mathsf{CF}$ for graph products M extends the question of when a graph product of groups is virtually free. The latter question has been answered completely by Lohrey and Sénizergues [**LohreySenizergues2007**]. A graph is called *chordal* if it does not contain an induced cycle of length $\geqslant 4$.

Theorem 6.3.2 (Lohrey, Sénizergues [**LohreySenizergues2007**])**.** *Let G_v be a finitely generated non-trivial group for each $v \in V$. Then $\mathbb{M}(\Gamma, (G_v)_{v\in V})$ is virtually free if and only if*

1. *for each $v \in V$, G_v is virtually free,*
2. *if G_v and G_w are infinite and $v \neq w$, then $\{v, w\} \notin E$,*
3. *if G_v is infinite, G_u and G_w are finite and $\{v, u\}, \{v, w\} \in E$, then $\{u, w\} \in E$, and*
4. *the graph Γ is chordal.*

6.4 Context-freeness for monoids

Our characterization of context-free monoids involves the equivalent conditions of Theorem 3.1.2. It is therefore convenient to assign these conditions a name.

Definition 6.4.1. *An* FRI-monoid *is a monoid that satisfies the equivalent conditions of Theorem 3.1.2. In other words, M is an FRI-monoid if and only if $\mathsf{R}_1(N)$ is finite for every finitely generated submonoid N of M.*

The first step in our characterization of context-free graph products is a description of context-free direct products. The following lemma could also be derived from a result of Latteux [**Latteux1979**], which states that if L_0, L_1 are languages over disjoint alphabets and $L_0 \sqcup\!\sqcup L_1$ is context-free, then one of the languages L_0, L_1 is regular.

Lemma 6.4.2. *The direct product of monoids M_0 and M_1 is context-free if and only if for some $i \in \{0,1\}$, M_i is context-free and M_{1-i} is an FRI-monoid.*

Proof. Suppose M_i is context-free and M_{1-i} is an FRI-monoid. Then each language $L \in \mathsf{VA}(M_i \times M_{1-i})$ is contained in $\mathsf{VA}(M_i \times N)$ for some finitely generated submonoid N of M_{1-i}. Since M_{1-i} is an FRI-monoid, N has finitely many right-invertible elements and hence $J_1(N)$ is a finite group (Corollary 3.2.3). Since no element outside of $J_1(N)$ can appear in a product yielding the identity, we may assume that $L \in \mathsf{VA}(M_i \times J_1(N))$. This means, however, that L can be accepted by a valence automaton over M_i by keeping the right component of the storage monoid in the state of the automaton. Hence, $L \in \mathsf{VA}(M_i)$ is context-free.

Suppose $\mathsf{VA}(M_0 \times M_1) \subseteq \mathsf{CF}$. Then certainly $\mathsf{VA}(M_i) \subseteq \mathsf{CF}$ for $i \in \{0,1\}$. This means we have to show that at least one of the monoids M_0 and M_1 is an FRI-monoid and thus, toward a contradiction, assume that none of them is.

By Corollary 3.2.3, for each i, there is a finitely generated submonoid N_i of M_i and infinite sets $S_0 \subseteq R_1(N_0)$ and $S_1 \subseteq L_1(N_1)$ such that the elements of S_0 have pairwise disjoint sets of right inverses in N_0 and the elements of S_1 have pairwise disjoint sets of left inverses in N_1. Let X_i be an alphabet large enough that we can find a surjective morphism $\varphi_i \colon X_i^* \to N_i$ for each $i \in \{0,1\}$. Furthermore, let $\#$ be a symbol with $\# \notin X_0 \cup X_1$. The language

$$L = \{r_0 \# r_1 \# s_0 \# s_1 \mid r_i, s_i \in X_i^*,\ \varphi_i(r_i s_i) = 1 \text{ for each } i \in \{0,1\}\}$$

is clearly contained in $\mathsf{VA}(M_0 \times M_1)$. We shall use Ogden's Lemma (Theorem 2.1.1) to show that L is not context-free. Suppose L is context-free and let m be the constant provided by Theorem 2.1.1. For each $a \in R_1(N_0)$, let $\ell_0(a)$ be the minimal length of a word $w \in X_0^*$ with $a\varphi_0(w) = 1$. Furthermore, for $a \in L_1(N_1)$, let $\ell_1(a)$ be the minimal length of a word $w \in X_1^*$ with $\varphi_1(w)a = 1$. The existence of the sets S_0 and S_1 guarantees that there are $a_0 \in R_1(N_0)$ and $a_1 \in L_1(N_1)$ such that $\ell_0(a_0) \geqslant m$ and $\ell_1(a_1) \geqslant m$. Choose $r_0 \in X_0^*$ and $s_1 \in X_1^*$ such that $\varphi_0(r_0) = a_0$ and $\varphi_1(s_1) = a_1$. Furthermore, let $r_1 \in X_1^*$ be a word of minimal length among those with $\varphi_1(r_1 s_1) = 1$ and let $s_0 \in X_0^*$ be a word of minimal length among those with $\varphi_0(r_0 s_0) = 1$. These choices guarantee $|r_1| \geqslant m$ and $|s_0| \geqslant m$. Moreover, the word $z = r_0 \# r_1 \# s_0 \# s_1$ belongs to L.

Let $z = uvwxy$ be the decomposition provided by the Ogden's Lemma, where we choose the positions in the subword $r_1 \# s_0$ to be marked. In the following, we call r_0, r_1, s_0, s_1 the *segments* of the word z. Clearly, v and x cannot contain the symbol $\#$. Therefore, by condition 2, at least one of the words v and x lies in one of the middle segments. By condition 3, they have to lie in the same segment or in neighboring segments. Hence, we have two cases:

- If v or x lies in the segment r_1, none of them lies in s_1. Thus, by pumping with $i = 0$, we obtain a word $r_0' \# r_1' \# s_0' \# s_1 \in L$ with $|r_1'| < |r_1|$ and $\varphi_1(r_1' s_1) = 1$, contradicting the choice of r_1.
- If v or x lies in the segment s_0, none of them lies in r_0. Thus, by pumping with $i = 0$, we obtain a word $r_0 \# r_1' \# s_0' \# s_1' \in L$ with $|s_0'| < |s_0|$ and $\varphi_1(r_0 s_0') = 1$, contradicting the choice of s_0.

This proves that L is not context-free and hence the lemma. □

In order to prove the main result of this chapter, we need a standard combinatorial fact about chordal graphs. A proof can be found, for example, in **[Diestel2010]**.

Proposition 6.4.3. *A simple graph is chordal if and only if it can be constructed recursively by pasting together graphs along complete subgraphs, starting with complete graphs.*

We are now ready to prove our main result on context-freeness. Note that Theorem 6.4.4 assumes that $J_1(M_v) \neq \{1\}$ for each $v \in V$. Let us explain why this is not a serious restriction. Since for a graph product $M = \mathbb{M}(\Gamma, (M_v)_{v\in V})$, there is a morphism $\varphi_v \colon M \to M_v$ for each $v \in V$ that restricts to the identity on M_v, we have $J_1(M) \cap M_v = J_1(M_v)$: While the inclusion "$\supseteq$" is true for any submonoid, given $b \in J_1(M) \cap M_v$ with $abc = 1$, $a, c \in M$, we also have $\varphi_v(a)b\varphi_v(c) = \varphi_v(abc) = 1$ and hence $b \in J_1(M_v)$. This means no element of $M_v \setminus J_1(M_v)$ can appear in a product yielding the identity. In particular, removing a vertex v with $J_1(M_v) = \{1\}$ will not change $\mathsf{VA}(M)$. Therefore, the requirement that $J_1(M_v) \neq \{1\}$ loses no explanatory power.

Theorem 6.4.4. *Let $\Gamma = (V, E)$ and let $J_1(M_v) \neq \{1\}$ for each $v \in V$. The monoid $M = \mathbb{M}(\Gamma, (M_v)_{v\in V})$ is context-free if and only if*

1. *for each $v \in V$, M_v is context-free,*
2. *if M_v and M_w are not FRI-monoids and $v \neq w$, then $\{v, w\} \notin E$,*
3. *if M_v is not an FRI-monoid, M_u and M_w are FRI-monoids and $\{v, u\}, \{v, w\} \in E$, then $\{u, w\} \in E$, and*
4. *the graph Γ is chordal.*

Proof. First, we show that conditions 1–4 are necessary. For 1, this is immediate and for 2, this is a consequence of Lemma 6.4.2. If 3 is violated, then for some $u, v, w \in V$, $M_v \times (M_u * M_w)$ is a submonoid of M such that M_u and M_w are FRI-monoids and M_v is not. Since M_u and M_w contain non-trivial (finite) subgroups, $M_u * M_w$ contains an infinite group and is thus not an FRI-monoid, meaning $M_v \times (M_u * M_w)$ is not context-free by Lemma 6.4.2.

Suppose 4 is violated for context-free M. By 2 and 3, any induced cycle of length at least four involves only vertices with FRI-monoids. Each of these, however, contains a non-trivial finite subgroup. This means M contains an induced cycle graph product of non-trivial finite groups, which is not virtually free by Theorem 6.3.2 and hence has a non-context-free identity language.

In order to prove the other direction, we note that $\mathsf{VA}(M) \subseteq \mathsf{CF}$ follows if $\mathsf{VA}(M') \subseteq \mathsf{CF}$ for every finitely generated submonoid $M' \subseteq M$. Since every such submonoid is contained in a graph product $N = \mathbb{M}(\Gamma, (N_v)_{v\in V})$ where each N_v is a finitely generated submonoid of M_v, it suffices to show that for such graph products, we have $\mathsf{VA}(N) \subseteq \mathsf{CF}$. This means whenever M_v is an FRI-monoid, N_v has finitely many right-invertible elements. Moreover, since $N_v \cap J_1(N) = J_1(N_v)$, no element of $N_v \setminus J_1(N_v)$ can appear in a product yielding the identity. Hence, if N_v is generated by $S \subseteq N_v$, replacing N_v by the submonoid generated by $S \cap J_1(N_v)$ does not change the identity languages of the graph product. Thus, we assume that each N_v is generated by a finite subset of $J_1(N_v)$. Therefore, whenever M_v is an FRI-monoid, N_v is a finite group.

If Γ is not connected, then N is the free product of the $\mathbb{M}(\Delta, (N_v)_{v\in V(\Delta)})$, where Δ ranges over the connected components of Γ. Since the connected components inherit conditions 1–4 and the free product preserves context-freeness (Theorem 2.6.3) we may assume that Γ is connected.

We observe furthermore that if S is a complete subgraph of Γ with $|V(S)| \geqslant 2$, then $N{\restriction_S}$ is a finite group: By condition 2 each N_v with $v \in V(S)$ has to be a finite group and $N{\restriction_S}$ is their direct product.

Since Γ is chordal, we can prove context-freeness of $\mathsf{VA}(N)$ by induction on the number of steps it takes to construct Γ by pasting together graphs along complete subgraphs (Proposition 6.4.3). In the induction base, we assume Γ is a complete graph. This means either $|V| = 1$ and N is context-free by condition 1 or $|V| \geqslant 2$ and N is a finite group and, in particular, context-free.

Now suppose that Γ is obtained by pasting together Γ_0 and Γ_1 along the complete subgraph S. We can clearly assume $\Gamma_0 \neq S$ and $\Gamma_1 \neq S$, because otherwise, Γ would coincide with Γ_0 or Γ_1. Since $N{\restriction_{\Gamma_0}}$ and $N{\restriction_{\Gamma_1}}$ are context-free by induction and $N \cong N{\restriction_{\Gamma_0}} *_{N{\restriction_S}} N{\restriction_{\Gamma_1}}$ (Lemma 6.2.1), context-freeness of N follows from Theorem 2.6.3 if we can show that $N{\restriction_S}$ is a finite group.

If $|V(S)| \geqslant 2$, we have observed this above. If $|V(S)| = 1$, say $V(S) = \{v\}$, the vertex v has neighbors $v_0 \in V(\Gamma_0)$ and $v_1 \in V(\Gamma_1)$, because otherwise, Γ would not be connected. By condition 3 and 2, this means $N_v = N{\restriction_S}$ is a finite group. □

Obstructions to context-freeness What does Theorem 6.4.4 tell us about features of storage mechanisms that obstruct context-freeness? As the proof demonstrates, conditions 2 and 3 forbid direct products of non-FRI-monoids. Furthermore, if conditions 2 and 3 are satisfied, condition 4 serves to rule out an induced cycle graph product of finite groups.

Direct products of non-FRI-monoids have an obvious interpretation as storage mechanisms: These are two independent non-trivial (in the sense that they admit non-regular languages) storage mechanisms. However, it appears to be hard to interpret induced cycle graph products of finite groups as storage mechanisms. In fact, it even seems difficult to find a language theoretic proof of non-context-freeness in this case. The proof given here relies on **LohreySenizergues2007**'s result. They provide two proofs of non-context-freeness, one of them uses virtual cohomology dimension and one uses Bass-Serre theory. Unfortunately, none of the two proofs yields an intuition why the resulting languages are not context-free.

As an example, consider the case that the graph is an induced cycle of length n and the factor groups are all isomorphic to $\mathbb{Z}/2\mathbb{Z}$ and each generated by some a_i for $i = 1, \ldots, n$. Then, the resulting identity language is the set of all words obtained from ε by (i) inserting $a_i a_i$ at some position and (ii) commuting a_i and a_{i+1} for $1 \leqslant i \leqslant n-1$ or commuting a_1 and a_n.

Reformulation We close this chapter by reformulating Theorem 6.4.4 in terms of expressive power of the factor monoids. Recall that $\mathsf{VA}(M) = \mathsf{Reg}$ if and only if M is an FRI-monoid (Theorem 3.1.2).

Corollary 6.4.5. *Let* $\Gamma = (V, E)$. *Then* $\mathsf{VA}(\mathbb{M}(\Gamma, (M_v)_{v \in V})) \subseteq \mathsf{CF}$ *if and only if*

1. *for each* $v \in V$, $\mathsf{VA}(M_v) \subseteq \mathsf{CF}$,
2. *if* $\mathsf{VA}(M_v) \neq \mathsf{Reg}$ *and* $\mathsf{VA}(M_w) \neq \mathsf{Reg}$ *and* $v \neq w$, *then* $\{v, w\} \notin E$,
3. *if* $\mathsf{VA}(M_v) \neq \mathsf{Reg}$, $\mathsf{VA}(M_u) = \mathsf{VA}(M_w) = \mathsf{Reg}$ *and* $\{v, u\} \in E$ *and* $\{v, w\} \in E$, *then* $\{u, w\} \in E$, *and*

4. *the graph* Γ *is chordal.*

6.5 Conclusion

In this chapter, we have studied which monoids M cause $\mathsf{VA}(M)$ to contain only context-free languages. Specifically, we have characterized those graph products of monoids that guarantee context-freeness.

The result identifies two obstructions for context-freeness. The first one consists of a direct product of two monoids that can each accept non-regular languages. In other words, two independently usable infinite storage mechanisms cause the languages to be non-context-free. The second obstruction is an induced cycle graph product (of length $\geqslant 4$) of finite groups. In this case, unfortunately, an intuition for the resulting storage mechanism seems difficult to obtain. In fact, for proving non-context-freeness, we relied on the result of **LohreySenizergues2007** [**LohreySenizergues2007**], who provided two proofs that are both group-theoretic.

Open problem We have seen in Lemma 6.4.2 that the context-free languages can only accommodate two independent storage mechanisms (i.e. include a class $\mathsf{VA}(M_0 \times M_1)$) if one of them is useless with respect to expressiveness (i.e. one of the monoids M_0, M_1 merely effects regular languages). This raises the question whether *building stacks* has this property in general: Is it true that if $\mathbb{B} * \mathbb{B} * M$ can accommodate the languages of $M_0 \times M_1$, then these are already accepted with M or one of the factors M_0, M_1 is useless? We conjecture that this is the case.

Conjecture 6.5.1. *Let* M, M_0, *and* M_1 *be monoids. Then*

$$\mathsf{VA}(M_0 \times M_1) \subseteq \mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$$

if and only if we have $\mathsf{VA}(M_0 \times M_1) \subseteq \mathsf{VA}(M)$ *or for some* $i \in \{0, 1\}$, *we have* $\mathsf{VA}(M_i) \subseteq \mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$ *and* $\mathsf{VA}(M_{1-i}) = \mathsf{VA}(\mathbf{1})$.

Note that the "if" direction holds trivially and that Lemma 6.4.2 is precisely the case $M = \mathbf{1}$. Observe also that $\mathsf{VA}(M_0 \times M_1) \subseteq \mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$ does not in general imply $\mathsf{VA}(M_i) \subseteq \mathsf{VA}(\mathbf{1})$ for some $i \in \{0, 1\}$, as in the case of $M = \mathbf{1}$. For example, if $M = \mathbb{Z} \times \mathbb{Z}$, then $\mathsf{VA}(\mathbb{Z} \times \mathbb{Z}) \subseteq \mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$, but $\mathsf{VA}(\mathbb{Z})$ contains non-regular languages. Therefore, we need to permit the case that $\mathsf{VA}(M_0 \times M_1) \subseteq \mathsf{VA}(M)$.

Related work As mentioned above, the question of which groups cause valence automata to accept only context-free languages is settled in a well-known result of Muller and Schupp [**MullerSchupp1983**] and Dunwoody [**Dunwoody1985**].

Our characterization of graph products that guarantee context-freeness extends a result of **LohreySenizergues2007** [**LohreySenizergues2007**]. In the positive branch, where context-freeness is shown, we provide an elementary language theoretic proof (employing Theorem 2.6.3), where **LohreySenizergues2007** rely on classic results on virtually free groups and thereby indirectly on the theorem of Muller, Schupp, and Dunwoody. In the negative branch, we use Lemma 6.4.2 in the case of direct products of non-FRI-monoids and we rely on **LohreySenizergues2007**'s proof in the case of induced cycle finite groups of length $\geqslant 4$.

The chapter also characterizes direct products that guarantee context-freeness (Lemma 6.4.2). Here, the non-context-freeness can alternatively be deduced from a result of **Latteux1979** [**Latteux1979**]. The latter states that for languages L_0 and L_1 over disjoint alphabets, if $L_0 \sqcup\!\sqcup L_1$ is context-free, then at least one of the languages L_0, L_1 is regular.

Acknowledgements I would like to thank Phoebe Buckheister for many discussions on valence automata.

Chapter 7

Semilinearity

7.1 Introduction

Parikh's theorem states that the Parikh image of every context-free language is effectively semilinear. This result is an extraordinarily useful tool, both for proving non-expressibility result and in the algorithmic analysis of formal languages. It has been extended to so many other language classes that the term 'a Parikh theorem' has come to mean a result guaranteeing effective semilinearity. This type of results has countless applications. Especially in cooperation with Presburger arithmetic, it facilitates a number of decision procedures.

Let us mention just a few examples from different areas. An early instance is **Ibarra1978**'s decision procedure for the emptiness problem of reversal bounded counter machines [**Ibarra1978**]. **HarjuIbarraKarhumakiSalomaa2002** [**HarjuIbarraKarhumakiSalomaa2002**] have used semilinearity to decide variants of Post's Correspondence Problem. A Parikh theorem has found application in in group theory, where **LohreySteinberg2008** [**LohreySteinberg2008**] used it in a decision procedure for the rational subset membership problem (see also Theorem 4.3.9). **SeidlSchwentickMuscholl2008** [**SeidlSchwentickMuscholl2008**] obtained algorithms for querying XML trees. Moreover, an application to timed automata has been proposed by **DaIbBuKeSu2000** [**DaIbBuKeSu2000**]. Furthermore, **BjoerklundBojanczyk2007** [**BjoerklundBojanczyk2007**] have used a Parikh theorem for deciding a fragment of first-order logic on words with nested data. For further applications, see [**KopczynskiTo2010**].

Therefore, understanding what storage mechanisms admit a Parikh theorem is useful for clarifying expressiveness, but especially in order to analyze automata. Hence, in this chapter, we study which monoids guarantee semilinearity of the accepted language class.

Our first result is a characterization of those graphs Γ for which $\mathsf{VA}(\mathbb{M}\Gamma)$ is semilinear. Since our characterization addresses all graph monoids, it generalizes Parikh's original theorem on context-free languages [**Parikh1966**] and the semilinearity of blind multicounter automata [**Ibarra1978**, **Greibach1978**]. Here, we present several equivalent conditions that capture different perspectives.

Our second result is that for each torsion group G, the class $\mathsf{VA}(\mathsf{G})$ contains only semilinear languages. While our first result provides effective semilinearity, the second cannot be effective in general, since there are torsion groups for which already the word problem is undecidable [**Adian2010**] and therefore, in particu-

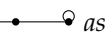
lar, emptiness of valence automata. We therefore also present a characterization of those torsion groups for which semilinear representations of Parikh images are computable.

Graph monoids One of the conditions in our desired characterization places those monoids with semilinearity in an inductively defined class. It is defined as follows. By SL, we denote the smallest isomorphism-closed class of monoids such that

1. $\mathbf{1} \in \mathsf{SL}$ and $\mathbb{B} \in \mathsf{SL}$ and
2. For each $M, N \in \mathsf{SL}$, we also have $M * N \in \mathsf{SL}$ and $M \times \mathbb{Z} \in \mathsf{SL}$.

Another notion that will appear is that of a transitive forest. We have already used it in Section 4.3 (Page 57), so that a definition can be found there. Our characterization for arbitrary graph monoids is the following.

Theorem 7.1.1. *For graphs Γ, the following conditions are equivalent:*

1. $\mathsf{VA}(\mathbb{M}\Gamma)$ *is semilinear.*
2. $\mathsf{VA}(\mathbb{B} \times \mathbb{B})$ *is not included in* $\mathsf{VA}(\mathbb{M}\Gamma)$.
3. (a) Γ^- *contains neither* C_4 *nor* P_4 *as an induced subgraph and*
   (b) Γ *contains neither* *nor*  *as an induced subgraph.*
4. (a) Γ^- *is a transitive forest and*
   (b) *the neighborhood of every unlooped vertex in* Γ *is a looped clique.*
5. $\mathbb{M}\Gamma \in \mathsf{SL}$.
6. $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{F}$.

Note that the condition 3b (and hence 4b) is similar to conditions 2 and 3 in Theorem 6.4.4 and Theorem 6.3.2: Instead of FRI-monoids (finite groups) we have looped vertices and instead of non-FRI-monoids (infinite groups), we have unlooped vertices.

A few comments on the equivalent conditions are in order. First, note that condition 2 describes the languages in $\mathsf{VA}(\mathbb{B} \times \mathbb{B})$ as obstructions to semilinearity. Hence, this explains precisely what property of the storage mechanism is responsible for non-semilinearity: *its ability to simulate two partially blind counters*. Second, condition 6 describes a common class containing all languages accepted by storage mechanisms with semilinearity. We will see in Proposition 7.1.2 that the semilinear storage mechanisms actually exhaust all of F.

Intuition on semilinear cases We also want to provide an automata-theoretic intuition of those storage mechanisms that guarantee semilinear Parikh images. As in Section 4.3, we do this by presenting a class of monoids that is equally expressive as SL. Let SC^- be the smallest isomorphism-closed class of monoids such that

1. $\mathbf{1} \in \mathsf{SC}^-$ and

2. for each $M \in \mathsf{SC}^-$, we also have $\mathbb{B} * M \in \mathsf{SC}^-$ and $M \times \mathbb{Z} \in \mathsf{SC}^-$.

Hence, the monoids in SC^- correspond to those storage mechanisms obtained from the trivial storage (**1**) by *building stacks* ($M \mapsto \mathbb{B} * M$) and *adding blind counters* ($M \mapsto M \times \mathbb{Z}$). Therefore, valence automata over monoids $M \in \mathsf{SC}^-$ are called *stacked counter automata*. For information on how to model recursive programs with numeric data types, see Chapter 12. In fact, we will not only show that stacked counter automata are as expressive as the monoids in SL, but also that the resulting languages are precisely those in F, yielding an explicit description of these languages:

Proposition 7.1.2. $\mathsf{VA}(\mathsf{SL}) = \mathsf{VA}(\mathsf{SC}^-) = \mathsf{F}$.

We will prove Theorem 7.1.1 and Proposition 7.1.2 in Section 7.2.

Torsion groups Our second result in this chapter concerns torsion groups. A group G is said to be a *torsion group* if for each element $g \in G$, there is a $k \in \mathbb{N} \setminus \{0\}$ with $g^k = 1$. Our result also makes reference to the rational subset membership problem, which is defined in Section 4.2.

Theorem 7.1.3. *Let G be a torsion group. Then every language in $\mathsf{VA}(G)$ is semilinear. Moreover, the Parikh images are computable for valence automata over G if and only if there is a uniform algorithm for the rational subset membership problem for all groups $G \times \mathbb{Z}^n$, $n \in \mathbb{N}$.*

One might wonder if a finitely generated torsion group is necessarily finite — which would make Theorem 7.1.3 quite boring. In fact, whether this is true was a long-standing open question in group theory, known as the *Burnside problem*. However, this is not the case: **GolodShafarevich1964** [**GolodShafarevich1964**, **Golod1964**] have shown that infinite such groups exist. Simple examples have been constructed by **Grigorchuk1980** [**Grigorchuk1980**] and by **GuptaSidki1983** [**GuptaSidki1983**].

Furthermore, there even exist finitely generated torsion groups with an undecidable word problem [**Adian2010**] and hence with an undecidable emptiness problem for valence automata. In particular, Parikh images of languages in $\mathsf{VA}(G)$ are not computable for every infinite torsion group G. Therefore, Theorem 7.1.3 provides a description—in terms of the rational subset membership problem—for which torsion groups the semilinearity is effective.

What makes torsion groups interesting to us is a result of **Render2010** [**Render2010**]. It implies that every language class $\mathsf{VA}(M)$ includes $\mathsf{VA}(\mathbb{B})$ or $\mathsf{VA}(\mathbb{Z})$, unless it is of the form $\mathsf{VA}(G)$ for a torsion group G. The latter case gives us little information about the language class. Therefore, Theorem 7.1.3 provides us with a language-theoretic condition in the otherwise somewhat unclear case of a torsion group. See Section 11.2 for details on **Render2010**'s result.

As another application of Theorem 7.1.3, we show that the semi-Dyck language over one pair of parentheses is not accepted by any valence automaton over $G \times \mathbb{Z}^n$, where G is a torsion group and $n \in \mathbb{N}$. Note that $D_1' \notin \mathsf{VA}(G)$ can be shown using a simple pumping argument that fails for the group $G \times \mathbb{Z}^n$.

Corollary 7.1.4. *For torsion groups G and $n \in \mathbb{N}$, we have $D_1' \notin \mathsf{VA}(G \times \mathbb{Z}^n)$.*

Proof. First, observe that $\mathsf{VA}(\mathbb{B} \times \mathbb{B})$ is not contained in $\mathsf{VA}(G \times \mathbb{Z}^n)$, since the former contains a non-semilinear language by Lemma 7.2.3 and the latter is semilinear by Theorem 7.1.3 and Proposition 2.5.3.

If D_1' were contained in $\mathsf{VA}(G \times \mathbb{Z}^n)$, then $\mathsf{VA}(\mathbb{B}) \subseteq \mathsf{VA}(G \times \mathbb{Z}^n)$, since D_1' is an identity language of $\mathbb{B}$. This implies

$$\mathsf{VA}(\mathbb{B} \times \mathbb{B}) \subseteq \mathsf{VA}((G \times \mathbb{Z}^n) \times (G \times \mathbb{Z}^n)) = \mathsf{VA}(G^2 \times \mathbb{Z}^{2n}),$$

contradicting our observation above, since G^2 is a torsion group as well. □

We will establish Theorem 7.1.3 in Section 7.3.
The results of this chapter have appeared in [**BuckheisterZetzsche2013a**].

7.2 Graph monoids

This section is devoted to the proof of Theorem 7.1.1 and Proposition 7.1.2. We begin with the latter.

Proof of Proposition 7.1.2. Since $\mathsf{SC}^- \subseteq \mathsf{SL}$, it suffices to prove the inclusions

$$\mathsf{VA}(\mathsf{SL}) \subseteq \mathsf{F} \subseteq \mathsf{VA}(\mathsf{SC}^-). \tag{7.1}$$

We begin with the left inclusion. We prove by induction with respect to the definition of SL that for each $M \in \mathsf{SL}$, we have $\mathsf{VA}(M) \subseteq \mathsf{F}_i$ for some $i \in \mathbb{N}$. The classes $\mathsf{VA}(\mathbf{1})$ and $\mathsf{VA}(\mathbb{B})$ are included in $\mathsf{CF} = \mathsf{G}_0 \subseteq \mathsf{F}_1$. Moreover, if $\mathsf{VA}(M), \mathsf{VA}(N) \subseteq \mathsf{F}_i$, then

$$\mathsf{VA}(M * N) \subseteq \mathsf{Alg}(\mathsf{VA}(M) \cup \mathsf{VA}(N)) \subseteq \mathsf{Alg}(\mathsf{F}_i) = \mathsf{G}_i \subseteq \mathsf{F}_{i+1}$$

by Theorem 2.6.3 and $\mathsf{VA}(M \times \mathbb{Z}) \subseteq \mathsf{SLI}(\mathsf{VA}(M)) \subseteq \mathsf{SLI}(\mathsf{F}_i) = \mathsf{F}_i$. This proves the left inclusion of Eq. (7.1).

We prove the right inclusion of Eq. (7.1) by induction on $i \in \mathbb{N}$: We show that $\mathsf{F}_i \subseteq \mathsf{VA}(\mathsf{SC}^-)$ for each $i \in \mathbb{N}$. For $i = 0$, we have $\mathsf{F}_0 \subseteq \mathsf{VA}(\mathbf{1})$, so suppose $i \geqslant 1$.

Recall that $\mathsf{F}_{i-1} \subseteq \mathsf{G}_{i-1} \subseteq \mathsf{F}_i$. We start with the inclusion $\mathsf{G}_{i-1} \subseteq \mathsf{VA}(\mathsf{SC}^-)$. For $L \in \mathsf{G}_{i-1}$, there is an F_{i-1}-grammar G with $L = L(G)$. Let $L_1, \ldots, L_n$ be the right-hand sides in G. By induction, for each right-hand side L_j, there is an $M_j \in \mathsf{SC}^-$ with $L_j \in \mathsf{VA}(M_j)$. By induction on the definition of SC^-, it is easy to see that for $M, N \in \mathsf{SC}^-$, there is a $P \in \mathsf{SC}^-$ such that $\mathsf{VA}(M) \subseteq \mathsf{VA}(P)$ and $\mathsf{VA}(N) \subseteq \mathsf{VA}(P)$. Hence, there is a $P \in \mathsf{SC}^-$ with $\mathsf{VA}(M_j) \subseteq \mathsf{VA}(P)$ for each j. Of course, we can choose P so that $P \neq \{1\}$. Then we have

$$L \in \mathsf{Alg}(\mathsf{VA}(M_1) \cup \cdots \cup \mathsf{VA}(M_n)) \subseteq \mathsf{Alg}(\mathsf{VA}(P)) \subseteq \mathsf{Alg}(\mathbb{B} * P),$$

in which the last inclusion is due to Theorem 2.6.6. Since $\mathbb{B} * P \in \mathsf{SC}^-$, this proves $L \in \mathsf{VA}(\mathsf{SC}^-)$.

Now suppose $L \in \mathsf{F}_i$. Then there is an $L' \in \mathsf{G}_{i-1}$, $L \subseteq X^*$, a homomorphism $h\colon X^* \to Y^*$ and a semilinear set $S \subseteq X^{\oplus}$ such that $L = h(L' \cap \Psi^{-1}(S))$. We have already established $\mathsf{G}_{i-1} \subseteq \mathsf{VA}(\mathsf{SC}^-)$, meaning there is an $M \in \mathsf{SC}^-$ with $L' \in \mathsf{VA}(M)$. The equation $L = h(L' \cap \Psi^{-1}(S))$ and Proposition 2.5.3 imply

$$L \in \mathsf{SLI}(\mathsf{VA}(M)) = \bigcup_{n \in \mathbb{N}} \mathsf{VA}(M \times \mathbb{Z}^n),$$

so that for some $n \in \mathbb{N}$, we have $L \in \mathsf{VA}(M \times \mathbb{Z}^n)$. Since $M \times \mathbb{Z}^n \in \mathsf{SC}^-$, this proves the right inclusion of Eq. (7.1). □

We continue with the proof of Theorem 7.1.1, specifically with the implication "1 ⇒ 2". The non-semilinear language we will exhibit can be traced back to the work of **Greibach1978 [Greibach1978]** and **Jantzen1979 [Jantzen1979]**. It is closely related to L_{bin}, which they used (independently) to show that in partially blind multicounter automata, silent transitions cannot be removed. L_{bin} will be used again in Section 8.5 to show that the same is true of automata with at least two partially blind counters and a number of blind counters.

Definition 7.2.1. *For $w \in \{0,1\}^*$, let $\mathrm{bin}(w)$ denote the number obtained by interpreting w as a base 2 representation:*

$$\mathrm{bin}(\varepsilon) = 0, \qquad \mathrm{bin}(w0) = 2 \cdot \mathrm{bin}(w), \qquad \mathrm{bin}(w1) = 2 \cdot \mathrm{bin}(w) + 1.$$

The language $L_{bin} \subseteq \{0,1,c\}^*$ is defined as

$$L_{bin} = \{wc^n \mid w \in \{0,1\}^*,\ n \leqslant \mathrm{bin}(w)\}.$$

Greibach1978 [Greibach1978] and **Jantzen1979 [Jantzen1979]** have shown the following.

Theorem 7.2.2 (Greibach1978 [Greibach1978] / Jantzen1979 [Jantzen1979]). *L_{bin} is accepted by an automaton with two partially blind counters. Hence, $L_{bin} \in \mathsf{VA}(\mathbb{B} \times \mathbb{B})$.*

This implies that the language $L_{bin} \cap \{1\}\{0,c\}^* = \{10^n c^m \mid m \leqslant 2^n\}$ also belongs to $\mathsf{VA}(\mathbb{B} \times \mathbb{B})$. The latter is clearly not semilinear. Indeed, if the set $S = \{(n,m) \in \mathbb{N}^2 \mid m \leqslant 2^n\}$ were Presburger definable, then so would be

$$\{\max\{m \in \mathbb{N} \mid (m,n) \in S\} \mid n \in \mathbb{N}\} = \{2^n \mid n \in \mathbb{N}\}.$$

This proves the next lemma.

Lemma 7.2.3. *$\mathsf{VA}(\mathbb{B} \times \mathbb{B})$ is not semilinear.*

The following lemma establishes the implication "2 ⇒ 3".

Lemma 7.2.4. *Suppose*

1. *Γ^- contains C_4 or P_4 as an induced subgraph or*
2. *Γ contains • — • or ◦ — • — ◦ as an induced subgraph.*

Then $\mathsf{VA}(\mathbb{B} \times \mathbb{B}) \subseteq \mathsf{VA}(\mathbb{M}\Gamma)$.

Proof. According to Theorem 4.3.1, if Γ^- contains C_4 or P_4 as an induced subgraph, then $\mathsf{VA}(\mathbb{M}\Gamma)$ contains all recursively enumerable languages and in particular $\mathsf{VA}(\mathbb{B} \times \mathbb{B})$.

If Γ contains • — • as an induced subgraph, then $\mathsf{VA}(\mathbb{B} \times \mathbb{B}) \subseteq \mathsf{VA}(\mathbb{M}\Gamma)$ is trivial. If Γ contains ◦ — • — ◦ as an induced subgraph, then $\mathbb{M}\Gamma$ contains a copy of $\mathbb{B} \times (\mathbb{Z} * \mathbb{Z})$ as a submonoid. By Theorem 2.4.1, we have $\mathsf{VA}(\mathbb{B}) \subseteq \mathsf{VA}(\mathbb{Z} * \mathbb{Z})$ and hence Corollary 2.3.7 implies $\mathsf{VA}(\mathbb{B} \times \mathbb{B}) \subseteq \mathsf{VA}(\mathbb{B} \times (\mathbb{Z} * \mathbb{Z}))$. □

If Γ^- contains C_4 as an induced subgraph, there is another way to see that $\mathsf{VA}(\mathbb{B} \times \mathbb{B}) \subseteq \mathsf{VA}(\mathbb{M}\Gamma)$: Since we have already shown that the presence of • — • or ◦ — • — ◦ as an induced subgraph guarantees $\mathsf{VA}(\mathbb{B} \times \mathbb{B}) \subseteq \mathsf{VA}(\mathbb{M}\Gamma)$, we may assume that all four participating vertices of Γ are looped. Hence, $\mathbb{M}\Gamma$ contains a

copy of $(\mathbb{Z} * \mathbb{Z}) \times (\mathbb{Z} * \mathbb{Z})$. By Theorem 2.4.1, we have $\mathsf{VA}(\mathbb{B}) \subseteq \mathsf{VA}(\mathbb{Z} * \mathbb{Z})$ and by Corollary 2.3.7, this implies $\mathsf{VA}(\mathbb{B} \times \mathbb{B}) \subseteq \mathsf{VA}(\mathbb{M}\Gamma)$.

The next lemma permits almost the same proof as Lemma 4.3.13 and yields the implication "4 $\Rightarrow$ 5".

Lemma 7.2.5. *Suppose that*

1. Γ^{-} *is a transitive forest and*
2. *the neighborhood of every unlooped vertex in* Γ *is a looped clique.*

Then $\mathbb{M}\Gamma \in \mathsf{SL}$.

Proof. Let $\Gamma = (V, E)$. We proceed by induction on $|V|$. Observe that by Theorem 4.3.12, every induced subgraph of a transitive forest is again a transitive forest. Therefore, every induced proper subgraph Δ of Γ satisfies 1 and 2 and our induction hypothesis implies $\mathbb{M}\Delta \in \mathsf{SL}$. If Γ is not connected, then Γ is the disjoint union of graphs Γ_1, Γ_2, for which $\mathbb{M}\Gamma_1, \mathbb{M}\Gamma_2 \in \mathsf{SL}$ by induction. Hence, $\mathbb{M}\Gamma \cong \mathbb{M}\Gamma_1 * \mathbb{M}\Gamma_2 \in \mathsf{SL}$. Therefore, we assume that Γ is connected.

Since Γ^{-} is a transitive forest, there is a vertex $v \in V$ that is adjacent to every vertex other than itself. We distinguish two cases.

- If v is a looped vertex, then $\mathbb{M}\Gamma \cong \mathbb{Z} \times \mathbb{M}(\Gamma \setminus v)$, and $\mathbb{M}(\Gamma \setminus \{v\}) \in \mathsf{SL}$ by induction.
- If v is an unlooped vertex, then by 2, $V \setminus \{v\}$ induces a looped clique. Thus, $\mathbb{M}\Gamma \cong \mathbb{B} \times \mathbb{Z}^{|V|-1} \in \mathsf{SL}$.

□

We are now ready to prove Theorem 7.1.1.

Proof of Theorem 7.1.1. Observe that condition 3b is equivalent to condition 4b. Therefore, Theorem 4.3.12 proves "3 $\Rightarrow$ 4" and we have

1 $\Longrightarrow$ 2	by Lemma 7.2.3
$\Longrightarrow$ 3	by Lemma 7.2.4
$\Longrightarrow$ 4	by Theorem 4.3.12
$\Longrightarrow$ 5	by Lemma 7.2.5
$\Longrightarrow$ 6	by Proposition 7.1.2
$\Longrightarrow$ 1	by Proposition 2.7.3

□

7.3 Torsion groups

This section is devoted to the proof of Theorem 7.1.3. The key ingredient is to show that a certain set of multisets is upward closed with respect to a well-quasi-ordering. Given multisets $\alpha, \beta \in X^{\oplus}$ and $k \in \mathbb{N}$, we write $\alpha \equiv_k \beta$ if k divides $\alpha(x) - \beta(x)$ for each $x \in X$. Furthermore, we write $\alpha \leqslant_k \beta$ if $\alpha \leqslant \beta$ and

$\alpha \equiv_k \beta$. Clearly, $\leqslant_k$ is a well-quasi-ordering on $X^\oplus$: Since $\equiv_k$ has finite index in $X^\oplus$, we find in any infinite sequence $\alpha_1, \alpha_2, \ldots \in X^\oplus$ an infinite subsequence $\alpha'_1, \alpha'_2, \ldots \in X^\oplus$ of $\equiv_k$-equivalent multisets. Since $\leqslant$ is a well-quasi-ordering, there are indices $i < j$ with $\alpha'_i \leqslant \alpha'_j$ and hence $\alpha'_i \leqslant_k \alpha'_j$.

If $S \subseteq X^\oplus$ is upward closed with respect to $\leqslant_k$, we also say S is k-*upward-closed*. The finite basis property of k-upward-closed sets then means that every k-upward-closed set is semilinear.

We begin with the proof of Theorem 7.1.3. Let G be a torsion group and K be accepted by the valence automaton $A = (Q, X, G, E, q_0, F)$. We may clearly assume that $F = \{f\}$ and $q_0 \neq f$. We regard the finite set E as an alphabet and define the automaton $\hat{A} = (Q, E, G, \hat{E}, q_0, F)$, where

$$\hat{E} = \{(p, (p, w, g, q), g, q) \mid (p, w, g, q) \in E\}.$$

Let $\hat{K} = L(\hat{A})$. Since K is a homomorphic image of $\hat{K}$, it suffices to show semilinearity (and computability) of $\hat{K}$. Since $q_0 \neq f$, we have $\varepsilon \notin \hat{K}$.

For a word $w \in E^*$, $w = (p_1, x_1, g_1, q_1) \cdots (p_n, x_n, g_n, q_n)$, we write $\sigma(w)$ for the set $\{p_i, q_i \mid 1 \leqslant i \leqslant n\}$. A non-empty word w is called a p, q-*computation* if $p_1 = p$, $q_n = q$, and $q_i = p_{i+1}$ for $1 \leqslant i < n$. A q, q-computation is also called a q-*loop*. Moreover, a q-loop w is called *simple* if $q_i \neq q_j$ for $i \neq j$.

For each subset $S \subseteq Q$, let F_S be the set of all q_0, f-computations $w \in E^*$ with $\sigma(w) = S$ and $|w| \leqslant |Q| \cdot (2^{|Q|} + 1)$. Furthermore, let $L_S \subseteq E^*$ consist of all $w \in E^*$ such that w is a simple q-loop for some $q \in S$ and $\sigma(w) \subseteq S$. Note that L_S is finite, which allows us to define the alphabet Y_S so as to be in bijection with L_S. Let $\varphi \colon Y_S \to L_S$ be this bijection and let $\tilde{\varphi} \colon Y_S^\oplus \to E^\oplus$ be the morphism satisfying $\tilde{\varphi}(y) = \Psi(\varphi(y))$ for $y \in Y_S$. For each $\nu \in F_S$, we define

$$U_\nu = \{\mu \in Y_S^\oplus \mid \exists w \in \hat{K} : \sigma(w) = \sigma(\nu),\ \Psi(w) = \Psi(\nu) + \tilde{\varphi}(\mu)\}$$

(note that there is only one $S \subseteq Q$ with $\nu \in F_S$) and claim that

$$\Psi(\hat{K}) = \bigcup_{S \subseteq Q} \bigcup_{\nu \in F_S} \Psi(\nu) + \tilde{\varphi}(U_\nu). \tag{7.2}$$

Lemma 7.3.1. *Equation (7.2) holds.*

Proof. The inclusion "$\supseteq$" holds by definition. For the other direction, we show by induction on n that for each q_0, f-computation $w \in E^*$, $|w| = n$, there is a $\nu \in F_S$ for $S = \sigma(w)$ and a $\mu \in Y_S^\oplus$ with $\sigma(w) = \sigma(\nu)$ and $\Psi(w) = \Psi(\nu) + \tilde{\varphi}(\mu)$. If $|w| \leqslant |Q| \cdot (2^{|Q|} + 1)$, this is satisfied by $\nu = w$ and $\mu = 0$. Therefore, assume $|w| > |Q| \cdot (2^{|Q|} + 1)$ and write $w = (p_1, x_1, g_1, q_1) \cdots (p_n, x_n, g_n, q_n)$. Since $n = |w| > |Q| \cdot (2^{|Q|} + 1)$, there is a $q \in Q$ that appears more than $2^{|Q|} + 1$ times in the sequence $q_1, \ldots, q_n$. Hence, we can write

$$w = w_0(p'_1, x'_1, g'_1, q)w_1 \cdots (p'_m, x'_m, g'_m, q)w_m$$

with $m > 2^{|Q|} + 1$. Observe that the word $w_i(p'_{i+1}, x'_{i+1}, g'_{i+1}, q)$ is a q-loop for each $1 \leqslant i < m$. Since $m - 1 > 2^{|Q|}$, there are indices $1 \leqslant i < j < m$ with

$$\sigma(w_i(p'_{i+1}, x'_{i+1}, g'_{i+1}, q)) = \sigma(w_j(p'_{j+1}, x'_{j+1}, g'_{j+1}, q)).$$

Now the word $w_i(p'_{i+1}, x'_{i+1}, g'_{i+1}, q)$ has as a factor a simple p-loop ℓ for some $p \in Q$. This means the word $w' \in E^*$, which is obtained from w by removing ℓ, satisfies $\sigma(w') = \sigma(w)$. Moreover, with $S = \sigma(w)$ and $\varphi(y) = \ell$, $y \in Y_S$, we have $\Psi(w) = \Psi(w') + \tilde{\varphi}(y)$. Finally, since $|w'| < |w|$, the induction hypothesis yields a $\nu \in F_S$ and a $\mu \in Y_S^{\oplus}$ with $\sigma(w') = \sigma(\nu)$ and $\Psi(w') = \Psi(\nu) + \tilde{\varphi}(\mu)$. Then we have $\sigma(w) = \sigma(\nu)$ and $\Psi(w) = \Psi(\nu) + \tilde{\varphi}(\mu + y)$ and the induction is complete.

In order to prove "$\subseteq$" of (7.2), suppose $w \in \hat{K}$. Since $\varepsilon \notin \hat{K}$, w is a q_0, f-computation and we can find the above $\nu \in F_S$, $S = \sigma(w)$, and $\mu \in Y_S^{\oplus}$ with $\sigma(w) = \sigma(\nu)$ and $\Psi(w) = \Psi(\nu) + \tilde{\varphi}(\mu)$. This means $\mu \in U_\nu$ and $\Psi(w)$ is contained in the right-hand side of (7.2). This proves (7.2). □

By (7.2) and since F_S is finite for each $S \subseteq Q$, for the semilinearity of $\Psi(\hat{K})$, it suffices to show that U_ν is semilinear for each $\nu \in F_S$ and $S \subseteq Q$. Let $\gamma\colon E^* \to G$ be the morphism with $\gamma((p, x, g, q)) = g$ for $(p, x, g, q) \in E$. Since G is a torsion group, the finiteness of L_S permits us to choose a $k \in \mathbb{N}$ such that $\gamma(\ell)^k = 1$ for any $\ell \in \bigcup_{S \subseteq Q} L_S$. We shall prove that U_ν is k-upward-closed.

Lemma 7.3.2. *For each $\nu \in F_S$, the set U_ν is k-upward-closed. In particular, $\Psi(\hat{K})$ is semilinear.*

Proof. It suffices to show that for $\mu \in U_\nu$, we also have $\mu + k \cdot y \in U_\nu$ for any $y \in Y_S$. Hence, let $\mu \in U_\nu$ with $w \in \hat{K}$ such that $\sigma(w) = \sigma(\nu)$ and $\Psi(w) = \Psi(\nu) + \tilde{\varphi}(\mu)$ and let $\mu' = \mu + k \cdot y$. Let $\ell = \varphi(y) \in L_S$ be a simple q-loop. Then $q \in S$ and since $\sigma(w) = \sigma(\nu) = S$, we can write $w = r(q_1, x_1, g_1, q)s$, $r, s \in E^*$. The fact that $w \in \hat{K}$ means in particular $\gamma(w) = 1$. Therefore, the word $w' = r(q_1, x_1, g_1, q)\ell^k s$ is a q_0, f-computation and satisfies $\gamma(w') = 1$ since $\gamma(\ell)^k = 1$. This means $w' \in \hat{K}$ and $\Psi(w') = \Psi(w) + k \cdot \Psi(\ell) = \Psi(\nu) + \tilde{\varphi}(\mu + k \cdot y)$. We also have $\sigma(\ell) \subseteq S$ and hence $\sigma(w') = \sigma(w) = \sigma(\nu)$. Therefore, the multiset $\mu' = \mu + k \cdot y$ belongs to U_ν. This proves U_ν to be k-upward-closed. □

We have thus proved the first statement of the theorem. The second statement requires two lemmas.

Lemma 7.3.3. *If Parikh images are computable for $\mathsf{VA}(G)$, then there is a uniform algorithm for the rational subset membership problem for all groups $G \times \mathbb{Z}^n$, $n \in \mathbb{N}$.*

Proof. Given a rational subset $R \subseteq G \times \mathbb{Z}^n$ and $g \in G \times \mathbb{Z}^n$, one can construct a valence automaton A over $G \times \mathbb{Z}^n$ such that $L(A) \neq \emptyset$ if and only if $g \in R$. According to Proposition 2.5.3, this allows us to compute a valence automaton A' over G, $L(A') \subseteq X^*$, a semilinear set $S \subseteq X^{\oplus}$, and a morphism h with $L(A) = h(L(A') \cap \Psi^{-1}(S))$. Since we can compute $\Psi(L(A'))$, we can decide whether $\Psi(L(A')) \cap S \neq \emptyset$, which is equivalent to $L(A) \neq \emptyset$. □

Lemma 7.3.4. *Given $\nu \in E^*$, one can construct a valence automaton A_ν over $G \times \mathbb{Z}^n$ such that $\Psi(L(A_\nu)) = U_\nu$.*

Proof. Consider the set

$$W_\nu = \{\Psi(w) - \Psi(\nu) \mid w \in \hat{K},\ \Psi(\nu) \leqslant \Psi(w),\ \sigma(w) = \sigma(\nu)\}$$

and observe that $U_\nu = \tilde{\varphi}^{-1}(W_\nu)$. Moreover, one can clearly construct a valence automaton A'_ν over G with $\Psi(L(A'_\nu)) = W_\nu$.

We now pick a linear order on E, which induces an embedding $E^{\oplus} \to \mathbb{Z}^n$ for $n = |E|$, by way of which we regard $E^{\oplus}$ as a subset of $\mathbb{Z}^n$. The new valence automaton A_ν over $G \times \mathbb{Z}^n$ works as follows. First, it simulates A'_ν. However, A_ν does not read any input during this simulation, but instead adds $\Psi(z)$ to the $\mathbb{Z}^n$-component of the storage group, where z is the input read by A'_ν. Afterwards, A_ν nondeterministically reads an input word $u \in Y_S^*$, subtracts $\tilde{\varphi}(\Psi(u))$ from the $\mathbb{Z}^n$-component, and goes into an accepting state. This means A_ν accepts u if and only if for some $z \in L(A'_\nu)$, we have $\tilde{\varphi}(\Psi(u)) = \Psi(z)$. Hence, $\Psi(L(A_\nu)) = \tilde{\varphi}^{-1}(W_\nu) = U_\nu$. □

It remains to be shown that $\Psi(\hat{K})$ is computable if there is a uniform algorithm for the rational subset problem for all groups $G \times \mathbb{Z}^n$, $n \in \mathbb{N}$.

Lemma 7.3.5. *Suppose that there is a uniform algorithm for the rational subset membership problem for groups $G \times \mathbb{Z}^n$, $n \in \mathbb{N}$. Then one can compute a semilinear representation of $\Psi(\hat{K})$.*

Proof. First, observe that since the rational subset membership problem is decidable for G, membership in the identity language is decidable as well. This means we can compute k. Moreover, by Theorem 4.2.1, given $\nu \in E^*$, we can decide whether $\hat{K} \cap \{\nu\} = \emptyset$ and thus compute F_S for each S. Hence, by (7.2), it suffices to compute a semilinear representation of U_ν.

Let A_ν be the automaton from Lemma 7.3.4. We will construct an ascending chain $V_0 \subseteq V_1 \subseteq \cdots$ of k-upward-closed subsets of U_ν. If $U_\nu \setminus V_i = \emptyset$, we stop; otherwise, we add new elements to obtain V_{i+1}. Since $\leqslant_k$ is a well-quasi-ordering, there is no infinite strictly ascending chain of k-upward-closed sets and the algorithm has to terminate. We initialize with $V_0 = \emptyset$. In order to check whether $U_\nu \setminus V_i = \emptyset$, we can similar to Corollary 2.8.3, construct a valence automaton A_i over $G \times \mathbb{Z}^n$ with $L(A_i) = L(A_\nu) \cap \Psi^{-1}(Y_S^{\oplus} \setminus V_i)$ (note that $\Psi^{-1}(Y_S^{\oplus} \setminus V_i)$ is effectively regular). Then clearly, $L(A_i) = \emptyset$ if and only if $U_\nu \setminus V_i = \emptyset$. Using the algorithm for the rational subset membership problem, we can decide whether $L(A_i) = \emptyset$ and hence whether $U_\nu \subseteq V_i$.

Should the inclusion $V_i \subseteq U_\nu$ still turn out to be strict, we enumerate all $\mu \in Y_S^{\oplus} \setminus V_i$ until we find a $\mu \in U_\nu$. The latter can again be checked by testing whether $L(A_\nu) \cap \Psi^{-1}(\mu) \neq \emptyset$. After finding μ, we set $V_{i+1} = (V_i \cup \{\mu\})\uparrow_k$. □

We have thus completed the proof of Theorem 7.1.3.

7.4 Conclusion

We have studied which storage mechanisms cause the accepted languages to be semilinear. In the first result, we have characterized those graph monoids that guarantee semilinearity.

The characterization provides several equivalent conditions from different perspectives. One condition tells us that the ability to simulate two partially blind counters is responsible for non-semilinearity; thus, the condition characterizes semilinearity in terms of the capabilities of the storage mechanism. Another condition states that a particular set of induced subgraphs may no occur, meaning that these are obstructions to semilinearity as induced subgraphs. Furthermore, we have seen that the languages accepted with storage mechanisms that

guarantee semilinearity are precisely those in the class F. Finally, we have identified a set of storage mechanisms that is expressively complete for F and therefore gives us an automata theoretic characterization of the languages in semilinear classes.

Our second result states that valence automata over torsion groups accept only semilinear languages. According to a result of **Render2010** [**Render2010**], this implies that every language class $\mathsf{VA}(M)$ contains $\mathsf{VA}(\mathbb{B})$ or $\mathsf{VA}(\mathbb{Z})$ or contains only semilinear languages. Furthermore, we have characterized those torsion groups for which semilinear representations of Parikh images are effectively computable.

Directions for future research Our first result completely explains semilinearity in the case of graph monoids. It seems prudent to try to understand better which groups have this property. It has been used by **LohreySteinberg2008** [**LohreySteinberg2008**] to solve the rational subset membership problem of graph groups (whenever it is decidable). In general, whenever we have effective semilinearity of $\mathsf{VA}(G)$, we can decide the rational subset membership problem, making semilinearity an interesting property of groups. We know so far that effective semilinearity is preserved by direct products with $\mathbb{Z}$ (Propositions 2.5.2 and 2.5.3) and by free products with amalgamation over a finite identified subgroup (Theorem 2.6.3). Furthermore, Theorem 7.1.3 tells us that torsion groups always enjoy this property and explains to some extent in what cases they do so effectively.

Related work In the case that all vertices of Γ are looped, $\mathbb{M}\Gamma$ is a graph group. Since **LohreySteinberg2008** [**LohreySteinberg2008**] characterized those graph groups that guarantee semilinearity, our result generalizes theirs. Our result also generalizes the fact that pushdowns and blind multicounter storages guarantee semilinearity.

As mentioned in Section 4.4, **MadhusudanParlato2011** [**MadhusudanParlato2011**] propose a general model of automata with auxiliary storage. Here, each storage mechanism is given by a class of graphs definable in monadic second order logic. They show that if this class of graphs has bounded tree-width, then the emptiness problem is decidable for automata with this type of storage mechanism. They also show that the multisets of vertex labels occurring in runs are semilinear in the case of bounded tree-width. Of course, the question arises whether Theorem 7.1.1 is subsumed by **MadhusudanParlato2011**'s.

Since the storage mechanisms in Section 7.2 almost always involve at least two counters, their configuration graphs (even when restricted to those from which a final configuration is reachable) can contain arbitrarily large grids. Moreover, the absence of arbitrarily large grids (as minors) is a characterizing property of bounded tree-width graph classes [**Diestel2010**]. It therefore seems unlikely that Theorem 7.1.1 is a special case of the framework of **MadhusudanParlato2011** [**MadhusudanParlat**

Acknowledgements I would like to thank Phoebe Buckheister for many discussions on semilinearity and valence automata.

Chapter 8

Silent transitions

8.1 Introduction

In the previous chapters, we measured the expressive power of various storage mechanisms with respect to what languages can be accepted by them in general valence automata. In this chapter, we study how this expressive power is affected by restricting the way in which the automata operate.

Specifically, we ask when silent transitions (which also called ε-transitions), i.e. those that read no input but may operate on the storage, are required to achieve the full expressive power.

This is an interesting problem for two reasons. First, it has consequences for the time and space complexity of the membership problem for these automata. For automata with silent transitions, it is in general not clear whether the membership problem is decidable. If, however, an automaton has no silent transitions, we only have to consider paths that are at most as long as the word at hand. In particular, if we can decide whether a sequence of storage operations is valid using linear space, we can also solve the membership problem (nondeterministically) with a linear space bound. Similarly, if we can decide validity of such a sequence in polynomial time, we can solve the membership problem in (nondeterministic) polynomial time. In fact, those storage mechanisms for which our results permit ε-removal always admit a linear space and an NP-algorithm for the membership problem.

Silent transitions Let $A = (Q, X, M, E, q_0, F)$ be a valence automaton over M. A transition $(p, w, m, q) \in E$ is called *silent transition* or *ε-transition* if $w = \varepsilon$, i.e. if it reads no input. The automaton A is said to be *ε-free* if it contains no ε-transitions. The class of languages accepted by ε-free valence automata is denoted by $\mathsf{VA}^+(M)$.

Note that we impose no restriction on the monoid elements that occur in an ε-free valence automaton. This means, if valence automata over M realize a concrete storage mechanism, then ε-free valence automata correspond to those where each input-reading transition is entitled to a bounded number of operations on the storage. This restriction is usually called *quasi-realtime* [**BookGreibach1970**].

As customary in this work, we want to present a class of monoids among which we characterize those that permit the removal of silent transitions. Clas-

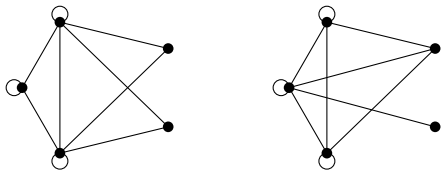


Figure 8.1: Examples of pseudo-bipartite graphs.

sical examples of storage mechanisms for which ε-transitions can be avoided are pushdown automata and blind multicounter automata, as shown by Greibach [**Greibach1965**, **Greibach1978**]. Our class of monoids that generalizes these two types is the class of $\mathbb{M}\Gamma$ where Γ is pseudo-bipartite.

Pseudo-bipartite graphs A graph is called *pseudo-bipartite* if

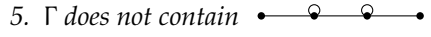
1. any two looped vertices are adjacent and
2. no two unlooped vertices are adjacent.

The term 'pseudo-bipartite' reflects the fact that in such graphs, the only choice we can make is which looped vertices are adjacent to which unlooped ones—the distribution of edges between looped vertices and between unlooped vertices is completely prescribed. In graph theory, a simple graph is called *bipartite* if its vertices can be divided into two disjoint sets A and B such that every edge connects a vertex from A with one in B [**Diestel2010**]. Hence, pseudo-bipartite graphs are obtained by choosing a number of looped vertices, a number of unlooped vertices, and a bipartite graph between the two sets of vertices.

Observe that looped cliques (which correspond to $\mathbb{Z}^n$) as well as graphs without any edges (which correspond to $\mathbb{B}^{(n)}$) are examples of pseudo-bipartite graphs. Hence, the class of pseudo-bipartite graphs generalizes pushdown storages and sets of blind counters. See Fig. 8.1 for further examples of pseudo-bipartite graphs.

Our first main result is the following. As usual, NP denotes the class of problems solvable by a nondeterministic algorithm in polynomial time [**Kozen1997**].

Theorem 8.1.1. *For pseudo-bipartite graphs Γ, the following conditions are equivalent:*

1. $\mathsf{VA}^+(\mathbb{M}\Gamma) = \mathsf{VA}(\mathbb{M}\Gamma)$.
2. *Every language in* $\mathsf{VA}(\mathbb{M}\Gamma)$ *is context-sensitive.*
3. *The membership problem of each language in* $\mathsf{VA}(\mathbb{M}\Gamma)$ *is in* NP.
4. *Every language in* $\mathsf{VA}(\mathbb{M}\Gamma)$ *is decidable.*
5. Γ *does not contain*  *as an induced subgraph.*
6. $\mathbb{M}\Gamma \in \mathsf{SC}^-$.

Since it is easy to see that each $M \in \mathsf{SC}^-$ can be written as $\mathbb{M}\Gamma$ for a pseudo-bipartite Γ, this implies that $\mathsf{VA}^+(M) = \mathsf{VA}(M)$ for each $M \in \mathsf{SC}^-$. Note also that together with Theorems 4.3.1 and 7.1.1, Theorem 8.1.1 implies that semilinearity of $\mathsf{VA}(\mathbb{M}\Gamma)$ is also equivalent to the condition of Theorem 8.1.1. Furthermore, since every language in F is accepted by stacked counter automaton (Proposition 7.1.2), the following is a consequence of Theorem 8.1.1.

Corollary 8.1.2. *Each language in* F *admits an* NP*-algorithm as well as a linear space algorithm for its membership problem.*

Theorem 8.1.1 generalizes several known results:

- An application of the Greibach normal form **[Greibach1965]** for context-free grammars is that ε-transitions can be avoided in pushdown automata.[1]
- Greibach has also shown that ε-transitions can be eliminated in blind multi-counter automata **[Greibach1978]**. While Greibach's construction triples the number of blind counters, an improved result by Latteux **[Latteux1979]** implies that this is not necessary. Put in terms of valence automata, this means $\mathsf{VA}^+(\mathbb{Z}^n) = \mathsf{VA}(\mathbb{Z}^n)$ for $n \in \mathbb{N}$. Since $\mathbb{Z}^n \in \mathsf{SC}^-$, this is subsumed by Theorem 8.1.1.
- Employing Latteux's result, **Hoogeboom2002 [Hoogeboom2002]** has shown that ε-transitions are avoidable in pushdown automata equipped with a set of blind counters. In other words, $\mathsf{VA}^+(\mathbb{B}^{(2)} \times \mathbb{Z}^n) = \mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{Z}^n)$ for $n \in \mathbb{N}$. This is a special case of Theorem 8.1.1 since $\mathbb{B}^{(2)} \times \mathbb{Z}^n \in \mathsf{SC}^-$. See Section 8.6 for a more detailed comparison of the proofs of Theorem 8.1.1 and Hoogeboom's result.

The author of this work learned of Latteux's and Hoogeboom's results after publication of **[Zetzsche2013a]**.

Extending the model of *Parikh automata* by **KlaedtkeRuess2003 [KlaedtkeRuess2003]** (see also Section 9.3.2), **Karianto2005 [Karianto2005]** studied *Parikh pushdown automata*. These are easily seen to be equivalent to automata with a pushdown and a set of blind counters. Since Theorem 8.1.1 implies $\mathsf{VA}^+(\mathbb{B}^{(2)} \times \mathbb{Z}^n) = \mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{Z}^n)$, we know in particular that Parikh pushdown automata also permit the removal of ε-transitions.

The simplest example of a storage mechanism in SC^- beyond the mentioned known examples with available ε-elimination is that consisting of one partially blind counter and a number of blind counters. This corresponds to the monoids $\mathbb{B} \times \mathbb{Z}^n$. (Note, however, that avoidability of ε-transitions does not become a weaker statement for simpler monoids.) Our second main result states that this the best we can do in mechanisms that combine a number of partially blind counters and a number of blind counters.

Theorem 8.1.3. *We have* $\mathsf{VA}^+(\mathbb{B}^r \times \mathbb{Z}^s) = \mathsf{VA}(\mathbb{B}^r \times \mathbb{Z}^s)$ *if and only if* $r \leqslant 1$.

In other words, when one has r partially blind counters and s blind counters, ε-transitions can be eliminated if and only if $r \leqslant 1$. In fact, our proof

[1] Strictly speaking and as mentioned before, our result only yields a quasi-realtime pushdown automaton. Syntactically, this is not quite the same as an ε-free pushdown automaton, since the latter can only pop one symbol at a time. However, by enlarging the stack alphabet, one can easily turn a quasi-realtime pushdown automaton into an an ε-free one.

will imply that the language in $\mathsf{VA}(\mathbb{B}^r \times \mathbb{Z}^s) \setminus \mathsf{VA}^+(\mathbb{B}^r \times \mathbb{Z}^s)$ is not contained in any $\mathsf{VA}^+(\mathbb{B}^p \times \mathbb{Z}^q)$ for $p, q \in \mathbb{N}$. Therefore, we generalize Greibach's and Jantzen's result that in partially blind multicounter automata, ε-transitions are indispensable. As mentioned above, since $\mathbb{B} \times \mathbb{Z}^s \in \mathsf{SC}^-$, the positive branch of Theorem 8.1.3 follows from Theorem 8.1.1. In order to prove the negative branch, we will extend the technique that both **Greibach1978** [**Greibach1978**] and **Jantzen1979** [**Jantzen1979**] used to show $\mathsf{VA}^+(\mathbb{B} \times \mathbb{B}) \subsetneq \mathsf{VA}(\mathbb{B} \times \mathbb{B})$. See Section 8.5.

Outline of the proofs The rest of this chapter is devoted to the proofs of Theorems 8.1.1 and 8.1.3. Let us give an outline and describe the main ingredients.

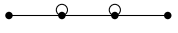
- In Section 8.2, we devote ourselves to decidability and complexity of the membership problem for ε-free valence automata. Here, we obtain a linear time and a (nondeterministic) polynomial time algorithm for the membership of each languages in $\mathsf{VA}^+(\mathbb{M}\Gamma)$, which establishes the implications "$1 \Rightarrow 2$" and "$1 \Rightarrow 3$" of Theorem 8.1.1. The algorithms are simple applications of convergent reduction systems.

- In Section 8.3, we develop a normal form result for rational subsets of monoids in SC^-. The underlying idea can be traced back to the work of **Benois1969** [**Benois1969**], who developed it to prove various results on rational sets of free groups.

  Such normal form results have been available for monoids described by monadic rewriting systems (see, for example, [**BookOtto1993**]) and we applied by **RenderKambites2009** [**RenderKambites2009**] to monoids representing pushdown storages. Under different terms, this normal form trick has been used by **BouajjaniEsparzaMaler1997** [**BouajjaniEsparzaMaler1997**] and by **Caucal2003** [**Caucal2003**] to describe rational sets of pushdown operations.

  However, since the monoids in SC^- allow commutation of non-trivial elements, an adaptation of this technique was necessary here. In the case of monadic rewriting systems, one transforms a finite automaton according to rewriting rules by gluing in new edges. Here, we glue in automata accepting languages that are Parikh equivalent to languages in $\mathsf{VA}(M)$ for $M \in \mathsf{SC}^-$.

  It should be mentioned that in [**Zetzsche2013a**] a slight variant of Proposition 8.3.1 was shown that subsumes the aforementioned normal form for pushdown automata. The one presented here is somewhat weaker but was used because it is simpler and makes the proofs more readable.

- We will see that those pseudo-bipartite graphs Γ without  as an induced subgraph define precisely the monoids in SC^-. Therefore, in Section 8.4, we prove ε-removal by induction on the construction of $M \in \mathsf{SC}^-$. This requires a stronger induction hypothesis: We show that it is not only possible to remove ε-transitions from valence automata, but even from valence transducers with output in a commutative monoid. Here, however, the constructed valence transducer is allowed to output a semilinear set in each step. Monoids that admit such a transformation will be called *strongly ε-independent*.

- Section 8.4 proceeds to show strong ε-independence of monoids in SC^- using three techniques.

  First, we show that $\mathbb{B}$ is strongly ε-independent (Section 8.4.1). Here, we use a construction that allows the postponement of increment operations and the early execution of decrement operations (Lemma 8.4.5). This is used to show that one can restrict oneself to computations in which a sequence of increments, followed by a sequence of decrements, will in the end change the counter only by a bounded amount.

  Second, we need one technique to show that if $M \in \mathsf{SC}^-$ is strongly ε-independent, then $M \times \mathbb{Z}$ is as well (Section 8.4.2). This uses semilinearity of $\mathsf{VA}(M)$ and an auxiliary lemma to provide small preimages for morphisms from the multisets to the integers (Lemma 8.4.6).

  The third technique is to show that building stacks preserves strong ε-independence, that is, if $M \in \mathsf{SC}^-$ is strongly ε-independent, where M is non-trivial, then $M * \mathbb{B}$ is as well (Section 8.4.3). The construction encodes rational sets over $M * \mathbb{B}$ as elements on the stack. We have to use the semilinearity of $\mathsf{VA}(M)$ again in order to be able to compute the set of all possible outputs when elements from two given rational sets cancel each other out.

- In Section 8.5, we prove Theorem 8.1.3. As mentioned above, since Theorem 8.1.1 already allows ε-removal for $\mathbb{B} \times \mathbb{Z}^n$, it remains to show that this is not possible for $\mathbb{B}^r \times \mathbb{Z}^s$ with $r \geqslant 2$. Here, we adapt a technique by **Greibach1978** [**Greibach1978**] and **Jantzen1979** [**Jantzen1979**] for showing that ε-transitions are indispensable in partially blind multicounter automata. Specifically, we use a concept from state complexity to reformulate and extend their argument.

The results of this chapter have appeared in [**Zetzsche2013a**].

8.2 The membership problem

In this section, we study decidability and complexity of the membership problem for valence automata over $\mathbb{M}\Gamma$ without silent transitions. Specifically, we show that for every Γ, membership for languages in $\mathsf{VA}^+(\mathbb{M}\Gamma)$ is (uniformly) decidable. We present two nondeterministic algorithms, one of them uses linear space and one runs in polynomial time (Proposition 8.2.4).

These results will serve two purposes. First, for those graphs Γ for which there are undecidable languages in $\mathsf{VA}(\mathbb{M}\Gamma)$ (see Theorem 4.3.1), it follows that silent transitions are indispensable. Second, if we can show that silent transitions can be removed from valence automata over $\mathbb{M}\Gamma$, the algorithms also apply to languages in $\mathsf{VA}(\mathbb{M}\Gamma)$.

8.2.1 A convergent reduction system

The algorithms in this section rely on the convergence property of certain reduction systems. For more information on reduction systems, see [**Huet1980**, **BookOtto1993**].

A *reduction system* is a pair $(S, \to)$ in which S is a set and $\to$ is a binary relation on S. $(S, \to)$ is said to be *noetherian* if there is no infinite sequence $s_0, s_1, \ldots$ with $s_i \to s_{i+1}$ for each $i \in \mathbb{N}$. We write $\overset{*}{\longleftrightarrow}$ ($\overset{*}{\to}$) for the reflexive, transitive, symmetric (reflexive, transitive) closure of $\to$. $(S, \to)$ has the *Church-Rosser property* if for any $s, t \in S$ with $s \overset{*}{\longleftrightarrow} t$, there is a $u \in S$ with $s \overset{*}{\to} u$ and $t \overset{*}{\to} u$. We say that $(S, \to)$ is *confluent*, if for any $s, t, u \in S$ with $s \overset{*}{\to} t$ and $s \overset{*}{\to} u$, there is a $v \in S$ with $t \overset{*}{\to} v$ and $u \overset{*}{\to} v$. A noetherian and confluent reduction system is called *convergent*. Furthermore, $(S, \to)$ is called *locally confluent*, if for any $s, t, u \in S$ with $s \to t$ and $s \to u$, there is a $v \in S$ with $t \overset{*}{\to} v$ and $u \overset{*}{\to} v$. An element $s \in S$ is *irreducible* if there is no $t \in S$ with $s \to t$. We say $t \in S$ is a *normal form* of $s \in S$ if $s \overset{*}{\to} t$ and t is irreducible. It is well-known that a reduction system is confluent if and only if it has the Church-Rosser property. Furthermore, a noetherian locally confluent reduction system is already confluent.

One of the steps in our algorithms will be to check, given a word $w \in X_\Gamma^*$, whether $w \equiv_\Gamma \varepsilon$. Unfortunately, turning the presentation T_Γ (see Section 2.4) into a reduction system on words will not yield a convergent reduction system as the length-preserving rules allow for infinite reduction sequences. Therefore, we will use reduction systems on traces instead. For more information on traces, see **[DiekertRozenberg1995]**.

Let X be an alphabet. An irreflexive symmetric relation $I \subseteq X \times X$ is called an *independence relation*. To each such relation, the corresponding presentation $\mathsf{T}_I = (X, R_I)$ is given as $R_I = \{(ab, ba) \mid (a, b) \in I\}$. If $\equiv_I$ denotes the congruence generated by T_I, then the monoid $\mathbb{T}(X, I) = X^*/\equiv_I$ is called *trace monoid*, its elements *traces*. The equivalence class of $u \in X^*$ is denoted as $[u]_I$ and since the words in an equivalence class all have the same length, $|[u]_I| = |u|$ is well-defined.

In order to efficiently compute using traces, we represent them using dependence graphs. Let X be an alphabet and $I \subseteq X \times X$ an independence relation. To each word $w \in X^*$ we assign a loop-free directed acyclic vertex-labeled graph, its *dependence graph* $\mathsf{dep}(w)$. If $w = x_1 \cdots x_n$, $x_i \in X$, $1 \leqslant i \leqslant n$, then $\mathsf{dep}(w) = (V, E, \ell)$, in which $E \subseteq V \times V$, has vertex set $V = \{1, \ldots, n\}$ and $(i, j) \in E$ if and only if $i < j$ and $(x_i, x_j) \notin I$. Furthermore, each vertex i is labeled with $\ell(i) = x_i \in X$. It is well-known that for words $u, v \in X^*$, we have $u \equiv_I v$ if and only if $\mathsf{dep}(u)$ and $\mathsf{dep}(v)$ are isomorphic. Thus, we will also write $\mathsf{dep}(s)$ for $\mathsf{dep}(u)$ if $s = [u]_I$.

Each (undirected, potentially looped) graph $\Gamma = (V, E)$ gives rise to an independence relation on X_Γ, namely

$$I = \{(x, y) \mid x \in \{a_v, \bar{a}_v\},\ y \in \{a_w, \bar{a}_w\},\ x \neq y,\ \{v, w\} \in E\}. \tag{8.1}$$

If I is given by Γ in this way, we also write $\equiv_{\Gamma|\mathbb{T}}$ for $\equiv_I$ and $[u]_{\Gamma|\mathbb{T}}$ instead of $[u]_I$.

In the following, let I be given by $\Gamma = (V, E)$ as in (8.1). We will now define a reduction relation $\to$ on $\mathbb{T}(X_\Gamma, I)$ such that for $u, v \in X_\Gamma^*$

$$[u]_\Gamma = [v]_\Gamma \quad \text{if and only if} \quad [u]_{\Gamma|\mathbb{T}} \overset{*}{\longleftrightarrow} [v]_{\Gamma|\mathbb{T}}. \tag{8.2}$$

For $s, t \in \mathbb{T}(X_\Gamma, I)$, let $s \to t$ if there are $u_1, u_2 \in X_\Gamma^*$ and $v \in V$ such that $s = [u_1 a_v \bar{a}_v u_2]_{\Gamma|\mathbb{T}}$ and $t = [u_1 u_2]_{\Gamma|\mathbb{T}}$. This definition immediately yields (8.2). Since our algorithms will represent traces as dependence graphs, we have to restate this relation in terms of the latter. It is not hard to see that for $s, t \in \mathbb{T}(X_\Gamma, I)$,

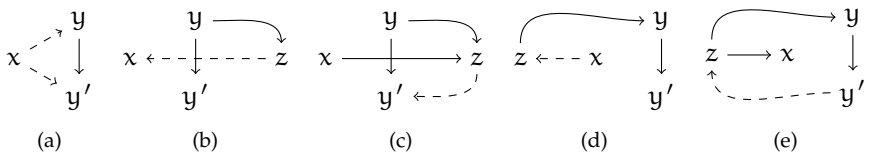


Figure 8.2: Possible fragments of the dependence graph of s.

$s \to t$ if and only if there are vertices x, y in $\mathsf{dep}(s)$, labeled a_ν and $\bar{a}_\nu$, respectively, such that

1. there is no path from y to x and
2. there is no vertex lying on a path from x to y

and $\mathsf{dep}(t)$ is obtained from $\mathsf{dep}(s)$ by deleting x and y. We will refer to conditions 1 and 2 as the *subtrace conditions*.

Lemma 8.2.1. *The reduction system* $(\mathbb{T}(X_\Gamma, I), \to)$ *is convergent.*

Proof. Since the system is clearly noetherian, it remains to be shown that the reduction system $(\mathbb{T}(X_\Gamma, I), \to)$ is locally confluent. Hence, let x, y, x', y' be vertices in $\mathsf{dep}(s)$ labeled $a_\nu, \bar{a}_\nu, a_w, \bar{a}_w$, respectively, satisfying the subtrace conditions such that $\mathsf{dep}(t)$ is obtained by deleting x, y and $\mathsf{dep}(t')$ is obtained by deleting x', y'. If $\{x, y\} = \{x', y'\}$, we are done. Furthermore, if $\{x, y\} \cap \{x', y'\} = \emptyset$, deleting x, y from $\mathsf{dep}(t')$ (or x', y' from $\mathsf{dep}(t)$) yields a $u \in \mathbb{T}(X_\Gamma, I)$ with $t \to u$ and $t' \to u$. Therefore, we assume $x = x'$ and $y \neq y'$ (the case $x \neq x'$, $y = y'$ can be done analogously). This means in particular that $\nu = w$. Since $(\bar{a}_\nu, \bar{a}_\nu) \notin I$, we can also assume that there is an edge from y to y'.

If $(a_\nu, \bar{a}_\nu) \notin I$, there are edges (x, y) and (x, y') in $\mathsf{dep}(s)$ and y violates the second subtrace condition of x, y' (see Fig. 8.2a). Hence, we have $(a_\nu, \bar{a}_\nu) \in I$. We claim that flipping y and y' constitutes an automorphism of $\mathsf{dep}(s)$, meaning $\mathsf{dep}(t)$ and $\mathsf{dep}(t')$ are isomorphic and thus $t = t'$. The former amounts to showing that each vertex z in $\mathsf{dep}(s)$ has an edge from (to) y iff z has one from (to) y'.

If there is an edge from y to z, then by the definition of I, we also have an edge between x and z. Obeying the first subtrace condition, it has to be directed from x to z: Otherwise, there would be a path from y to x (see Fig. 8.2b). Since y and y' share the same label, we also have an edge between y' and z. If this were an edge from z to y', z would lie on a path from $x = x'$ to y' (see Fig. 8.2c), violating the second subtrace condition. Hence, there is an edge from y' to z.

If there is an edge from z to y, then by the definition of I, we also have an edge between x and z. By the second subtrace condition, it has to be directed from z to x: Otherwise, z would lie on a path from x to y (see Fig. 8.2d). Since y and y' share the same label, we also have an edge between y' and z. If this were directed from y' to z, then there would be a path from y' to $x = x'$ (see Fig. 8.2e), violating the first subtrace condition. Hence, there is an edge from z to y'.

If there is no edge between y and z, there is also no edge between y' and z, since y and y' have the same label. □

By (8.2) and since $(\mathbb{T}(X_\Gamma, I), \to)$ is convergent, we have

$$[w]_\Gamma = [\varepsilon]_\Gamma \text{ if and only if } [w]_{\Gamma|\mathbb{T}} \xrightarrow{*} [\varepsilon]_{\Gamma|\mathbb{T}}. \tag{8.3}$$

This equivalence is the basis of our algorithms to check for the former condition.

8.2.2 Decidability and complexity

The following lemmas provide algorithms for the identity problem of $\mathbb{M}\Gamma$, which asks, given $w \in X_\Gamma^*$, whether $w \equiv_\Gamma \varepsilon$. One of the algorithms requires polynomial time and the other one linear space. In fact, there are classes of graphs Γ that admit more efficient algorithms.

- Suppose every vertex in Γ is looped. Then $\mathbb{M}\Gamma$ belongs to the class of graph groups, which are linear over $\mathbb{R}$ **[HsuWise1999, DavisJanuszkiewicz2000]** (that is, they embed into some $\mathsf{GL}(n, \mathbb{R})$ with $n \in \mathbb{N}$). Since **LiptonZalcstein1977 [LiptonZalcstei** have shown that the word problem of groups that are linear over a field of characteristic zero is decidable in deterministic logarithmic space, we know the same for $\mathbb{M}\Gamma$.

  Moreover, the word problem for graph groups can be solved in linear time, which as proved by **Wrathall1988 [Wrathall1988]**.

- If Γ contains no edges at all, then $\mathbb{M}\Gamma \cong \mathbb{B}^{(n)}$ for $n = |\Gamma|$ and the set of words w with $w \equiv_\Gamma \varepsilon$ is a semi-Dyck language over n pairs of parentheses, which can be recognized in deterministic logspace as shown by **RitchieSpringsteel1972 [RitchieSpringsteel1972]**.

- If Γ contains no loops, but any two distinct vertices are adjacent in Γ, then $\mathbb{M}\Gamma \cong \mathbb{B}^n$ for $n = |\Gamma|$ and we can decide whether $w \equiv_\Gamma \varepsilon$ by using n partially blind counters, which take up only logarithmic space.

Therefore, it seems likely that the following algorithms for the identity problem can be improved. However, their purpose here is to facilitate a linear space and an NP algorithm for the membership problem for languages in $\mathsf{VA}^+(\mathbb{M}\Gamma)$ (Proposition 8.2.4). Our solution for the latter problem, however, still involves guessing a run of an automaton, meaning that the algorithms for the membership problem would not profit from improvements on the identity problem. In fact, the membership problem is in some cases NP-complete, so that at least the NP upper bound would certainly not be affected (see Proposition 8.2.4).

Lemma 8.2.2. *There is a deterministic polynomial-time algorithm that, given a word $w \in X_\Gamma^*$, determines whether $[w]_\Gamma = [\varepsilon]_\Gamma$.*

Proof. By (8.3), the condition $[w]_\Gamma = [\varepsilon]_\Gamma$ is equivalent to $[\varepsilon]_{\Gamma|\mathbb{T}}$ being the normal form of $[w]_{\Gamma|\mathbb{T}}$. Therefore, our algorithm computes the normal form of $[w]_{\Gamma|\mathbb{T}}$. It does so by computing the dependence graph of w and successively deleting pairs of nodes that satisfy the subtrace conditions. Finding such a pair can be done in polynomial time and since at most $|w|/2$ deletions are possible, the normal form is computed in polynomial time. In the end, the algorithm checks whether the calculated dependence graph representing the normal form is empty. □

Lemma 8.2.3. *There is a nondeterministic linear-space algorithm that, given a word $w \in X_\Gamma^*$, determines whether $[w]_\Gamma = [\varepsilon]_\Gamma$.*

Proof. Let $\Gamma = (V, E)$. By (8.3), we have $[w]_\Gamma = [\varepsilon]_\Gamma$ if and only if w can be reduced to the empty word by commuting a_v and a_w for $v, w \in E$, commuting a_v and $\bar{a}_v$ for $\{v\} \in E$ and deleting $a_v \bar{a}_v$ for $v \in V$. This can be done by a nondeterministic linear-space algorithm. □

The following proposition uses the algorithms for the identity problem of $\mathbb{M}\Gamma$ to devise a procedure for the membership problem of languages in $\mathsf{VA}^+(\mathbb{M}\Gamma)$. The NP-hardness follows easily from a result of **BookGreibach1970 [BookGreibach1970]** stating that every language accepted by a linear time nondeterministic Turing machine is a length-preserving homomorphic image of the intersection of three context-free languages.

Proposition 8.2.4. *For each $L \in \mathsf{VA}^+(\mathbb{M}\Gamma)$, the membership problem can be decided by*

- *a nondeterministic polynomial-time algorithm as well as*
- *a nondeterministic linear-space algorithm.*

In particular, the languages in $\mathsf{VA}^+(\mathbb{M}\Gamma)$ are context-sensitive. Moreover, there is a graph Γ and a language $L \in \mathsf{VA}^+(\mathbb{M}\Gamma)$ such that membership in L is NP-complete.

Proof. In order to decide the membership problem for a word w for a language in $\mathsf{VA}^+(\mathbb{M}\Gamma)$, we can guess a run reading w. Since there are no silent transitions in the automaton, such a run has length linear in $|w|$. For this run, we have to check whether the product of the monoid elements on the edges is the identity element of $\mathbb{M}\Gamma$. By Lemmas 8.2.2 and 8.2.3, this can be done in polynomial time or using linear space.

We now construct Γ and $L \in \mathsf{VA}^+(\mathbb{M}\Gamma)$ such that L has an NP-complete membership problem. Pick a nondeterministic linear time Turing machine that accepts an NP-complete language L (the linear time bound can be achieved by a suitable padding of the input). According to **[BookGreibach1970]**, each language accepted by such a Turing machine can be written as a length-preserving homomorphic image of the intersection of three context-free languages. If we choose Γ such that $\mathbb{M}\Gamma \cong (\mathbb{B} * \mathbb{B})^3$, then $\mathsf{VA}^+(\mathbb{M}\Gamma)$ contains L. □

8.3 Rational sets

In this section and the next, we use some new notation. If $A = (Q, M, E, q_0, F)$ is an automaton over M, then we write

$$L_{p,q}(A) = \{m \in M \mid (p, 1) \to_A^* (q, m)\}.$$

In other words, $L_{p,q}(A)$ contains those elements of M that labels paths from p to q. In the following proofs, we will construct automata by gluing in some automaton between two states of another. Let us define this formally. Suppose that $A = (Q, M, E, q_0, F)$ is an automaton over M and $B = (Q', M, E', q_0', \{q_f'\})$ is an automaton over M with only one final state. Suppose furthermore that $Q \cap Q' = \emptyset$. Then the automaton obtained by *gluing in* B *between* $p, q \in Q$ is defined as $C = (Q \cup Q', M, E'', q_0, F)$, where

$$E'' = E \cup E' \cup \{(p, 1, q_0'), (q, 1, q_f')\}.$$

The following normal form result is in the spirit of a well-known fact about monadic string rewriting systems [**BookOtto1993**]. The latter states that for each rational subset of a monoid described by such a rewriting system, one can construct a regular language that consists solely of normal forms of elements, but represents every member of the rational subset.

The construction goes back to an idea of **Benois1969** [**Benois1969**] and works by successively adding edges to obtain more representatives for the rational set. Here, convergence is guaranteed because only edges are added, not new states. Since in our monoids, non-trivial elements can commute, we are not in the situation of a monadic rewriting system, meaning that adding edges does not suffice. Therefore, we glue in automata instead of edges and show that termination is still guaranteed because we only have to glue in one automaton between each pair of states.

Recall that a and $\bar{a}$ denote the positive and the negative generator, respectively, of the bicyclic monoid $\mathbb{B}$ (see Section 2.2, Page 14).

Proposition 8.3.1. *Let $M \in \mathsf{SC}^-$ and C be a commutative monoid. Furthermore, let $S \subseteq (M * \mathbb{B}) \times C$ be a rational set. Then there is an alphabet $X = \{x, \bar{x}\} \cup Y \cup Z$ and a morphism $\varphi\colon X^* \to (M * \mathbb{B}) \times C$ such that $\varphi(x) = a$, $\varphi(\bar{x}) = \bar{a}$, $\varphi(Y) \subseteq M$, $\varphi(Z) \subseteq C$, and regular languages*

$$U_i \subseteq ((Y \cup Z)^* \bar{x})^*, \qquad V_i \subseteq (Y \cup Z)^*, \qquad W_i \subseteq (x(Y \cup Z)^*)^*.$$

such that

$$S \cap (J_1(M * \mathbb{B}) \times C) = \bigcup_{i=1}^{n} \varphi(U_i V_i W_i) \cap (J_1(M * \mathbb{B}) \times C).$$

Proof. Let $S \subseteq (M * \mathbb{B}) \times C$ be rational. Then there is an alphabet X, a rational language $L \subseteq X^*$, and a morphism $\varphi\colon X^* \to (M * \mathbb{B}) \times C$ with $\varphi(L) = S$. Without loss of generality, we assume that $X = \{x, \bar{x}\} \cup Y \cup Z$ with $\varphi(x) = a$, $\varphi(\bar{x}) = \bar{a}$, $\varphi(Y) \subseteq M$, $\varphi(Z) \subseteq C$, where a and $\bar{a}$ are the two generators of $\mathbb{B}$. Let A be an automaton accepting L such that every edge carries exactly one letter.

As a first step, we will construct an automaton A' that also satisfies the equation $\varphi(L(A')) = S$, but which represents every element of $S \cap (J_1(M * \mathbb{B}) \times C)$ by a word in $X^* \setminus X^* x X^* \bar{x} X^*$. Let $A = (Q, X, E, q_0, F)$. For $p, q \in Q$, the language

$$K_{p,q} = \{\pi_Z(w) \mid w \in L_{p,q}(A),\ \varphi(w) \in \{1\} \times C\}$$

is clearly contained in $\mathsf{VA}(M * \mathbb{B})$ and is therefore semilinear (Theorem 7.1.1 and Proposition 7.1.2). Thus, we can find a finite automaton $A'_{p,q}$ such that $\Psi(L(A'_{p,q})) = \Psi(K_{p,q})$. Since C is commutative and $\varphi(Z) \subseteq C$, this also means $\varphi(L(A'_{p,q})) = \varphi(K_{p,q})$. The automaton A' is now obtained from A by gluing $A'_{p,q}$ into A between p and q, for each $p, q \in Q$. Since in A' for each path from the initial to the final state, we can find another path that encodes the same element of $(M * \mathbb{B}) \times C$ and is present in A, we have $\varphi(L(A')) = \varphi(L(A)) = S$. However, the glued in automata allow us to encode elements of the intersection $S \cap (J_1(M * \mathbb{B}) \times C)$ by words of a certain form: We claim that

$$S \cap (J_1(M * \mathbb{B}) \times C) \subseteq \varphi(L(A') \setminus X^* x X^* \bar{x} X^*) \subseteq S. \tag{8.4}$$

The right inclusion is clear because $\varphi(L(A')) = S$. For the left inclusion, let $s \in S \cap (J_1(M * \mathbb{B}) \times C)$ and $w \in L(A')$ be chosen such that $\varphi(w) = s$ and $|w|_{X \setminus Z}$

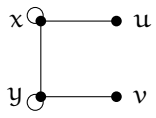


Figure 8.3: Induced subgraph in the proof of Lemma 8.4.1.

is minimal. Toward a contradiction, suppose $w \in X^*xX^*\bar{x}X^*$. Then $w = fxg\bar{x}h$ with $f, h \in X^*$, $g \in (Y \cup Z)^*$. Since $am\bar{a} \notin J_1(M * \mathbb{B})$ for any $m \in M \setminus \{1\}$, our assumption $s \in J_1(M * \mathbb{B}) \times C$ implies $\varphi(g) \in \{1\} \times C$ and thus $\varphi(xg\bar{x}) \in \{1\} \times C$. By the construction of A', however, this means that there is a word $v \in Z^*$ such that $fvh \in L(A')$ and $\varphi(fvh) = \varphi(w)$. Since $|fvh|_{X\setminus Z} = |fh|_{X\setminus Z} < |w|_{X\setminus Z}$, this contradicts the choice of w, proving Eq. (8.4).

Since the language $K = L(A') \setminus X^*xX^*\bar{x}X^*$ satisfies

$$K \subseteq ((Y \cup Z)^*\bar{x})^*(Y \cup Z)^*(x(Y \cup Z)^*)^*,$$

we can find regular languages

$$U_i \subseteq ((Y \cup Z)^*\bar{x})^*, \qquad V_i \subseteq (Y \cup Z)^*, \qquad W_i \subseteq (x(Y \cup Z)^*)^*$$

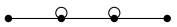
for $1 \leqslant i \leqslant n$ with $K = \bigcup_{i=1}^{n} U_iV_iW_i$. Since Eq. (8.4) implies the equality

$$\varphi(K) \cap (J_1(M * \mathbb{B}) \times C) = S \cap (J_1(M * \mathbb{B}) \times C),$$

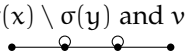
this is the desired decomposition. □

8.4 Stacked blind counters

In this section, we prove the implications "5 ⇒ 6" and "6 ⇒ 1" of Theorem 8.1.1. We begin with the former.

Lemma 8.4.1. *Let Γ be a pseudo-bipartite graph that does not contain*  *as an induced subgraph. Then, $\mathbb{M}\Gamma \in \mathsf{SC}^-$.*

Proof. We proceed by induction on the number of vertices. Note that induced subgraphs of Γ are also pseudo-bipartite and do not contain  . Therefore, we may assume that $\mathbb{M}(\Gamma \setminus x) \in \mathsf{SC}^-$ for any vertex x. Let $\Gamma = (V, E)$ and write $V = L \cup U$, where L is the set of looped vertices and U is the set of unlooped vertices. For every $x \in L$, let $\sigma(x) = N(x) \cap U$ be the set of unlooped neighbors of x. We write $x \leqslant y$ for $x, y \in L$ if $\sigma(x) \subseteq \sigma(y)$. Clearly, $\leqslant$ is a reflexive, transitive order on L.

If there were $x, y \in L$ such that $\sigma(x)$ and $\sigma(y)$ are incomparable, there would be vertices $u, v \in U$ with $u \in \sigma(x) \setminus \sigma(y)$ and $v \in \sigma(y) \setminus \sigma(x)$. Thus, the vertices u, x, y, v induce the subgraph  (see Fig. 8.3), contradicting our premise. Hence, $\leqslant$ is a total order and has a greatest element $g \in L$, that is, such that $x \leqslant g$ for each $x \in L$ (note that g may not be unique as $\leqslant$ is not necessarily antisymmetric).

- If $\sigma(g) = U$, then g is adjacent to every vertex in Γ and thus we have $\mathbb{M}\Gamma \cong \mathbb{M}(\Gamma \setminus g) \times \mathbb{Z}$.

- If $\sigma(g) \subsetneq U$, then there is an isolated vertex $u \in U \setminus \sigma(g)$. Hence, we have $\mathbb{M}\Gamma \cong \mathbb{M}(\Gamma \setminus u) * \mathbb{B}$.

□

As an example for the previous lemma, consider the graphs Γ_a and Γ_b in Figs. 8.1a and 8.1b. Then, we have

$$\mathbb{M}\Gamma_a \cong ((((\mathbf{1} \times \mathbb{Z}) * \mathbb{B}) * \mathbb{B}) \times \mathbb{Z}) \times \mathbb{Z},$$
$$\mathbb{M}\Gamma_b \cong (((\mathbb{B} \times \mathbb{Z}) \times \mathbb{Z}) * \mathbb{B}) \times \mathbb{Z}.$$

The core of Theorem 8.1.1 is the fact that $\mathsf{VA}^+(M) = \mathsf{VA}(M)$ for every monoid $M \in \mathsf{SC}^-$. We prove this using an induction with respect to the definition of SC^-. In order for this induction to work, we need to strengthen the induction hypothesis. This stronger hypothesis will state that we can not only eliminate ε-transitions, but we can do this while preserving the output in a commutative monoid. More precisely, for any $M \in \mathsf{SC}^-$ and any commutative monoid C, we can transform a valence transducer over M with output in C into another one that has no ε-transitions but is allowed to output a semilinear set of elements in each step. Formally, we will show that each $M \in \mathsf{SC}^-$ is strongly ε-independent.

Valence transducers with commutative output In the following, we extend the definition of valence transducers (Section 3.1) so as to allow outputs in arbitrary monoids (as opposed to words). It will therefore be useful to make the output monoid of a transduction $T \subseteq X^* \times M$ explicit by called it an *M-transduction*. Let M and N be monoids. A *valence transducer over M with output in N* is an automaton over $X^* \times M \times N$. Instead of $(Q, X^* \times M \times N, E, q_0, F)$, we also write $A = (Q, X, M, N, E, q_0, F)$. Such devices perform N-transductions, namely

$$T(A) = \{(u, v) \in X^* \times N \mid \exists q \in F\colon (q_0, \varepsilon, 1, 1) \to_A^* (q, u, 1, v)\}.$$

The class of N-transductions performed by valence transducers over M is denoted by $\mathsf{VT}(M, N)$. An edge of the form (ε, m, n) with $n \in N$, $m \in M$ is called *ε-transition* and A is *ε-free* if such transitions are absent. By confining ourselves to ε-free transducers, we obtain the class $\mathsf{VT}^+(M, N)$ of N-transductions.

As mentioned above, we want to show that we can remove ε-transitions while preserving the output in a commutative monoid C. Note that we cannot hope to prove $\mathsf{VT}^+(M, C) = \mathsf{VT}(M, C)$, since valence transducers without ε-transitions and with output in C can only output finitely many elements per input word. However, we will show that if we grant the ε-free transducers the ability to output a semilinear set in one step, they are expressively complete (with respect to C-transductions). Let us formalize this property.

Let C be a commutative monoid and $T \subseteq X^* \times \mathsf{SL}(C)$ be an $\mathsf{SL}(C)$-transduction. Then $\Phi(T) \subseteq X^* \times C$ is defined as

$$\Phi(T) = \{(w, c) \in X^* \times C \mid \exists (w, S) \in T\colon c \in S\}.$$

For a class $\mathcal{C}$ of $\mathsf{SL}(C)$-transductions, $\Phi(\mathcal{C})$ is the class of all $\Phi(T)$ with $T \in \mathcal{C}$. A monoid M is called *strongly ε-independent* if for every commutative monoid C, we have

$$\mathsf{VT}(M, C) = \Phi(\mathsf{VT}^+(M, \mathsf{SL}(C))).$$

By choosing the trivial monoid for C, we can see that for every strongly ε-independent monoid M, we have $\mathsf{VA}^+(M) = \mathsf{VA}(M)$. Indeed, given a valence automaton A over M, add an output of 1 to each edge and transform the resulting valence transducer into an ε-free one with output in $\mathsf{SL}(\mathbf{1})$. The latter can then clearly be turned into a valence automaton for the language accepted by A. Hence, we have the following.

Lemma 8.4.2. *If M is strongly ε-independent, then $\mathsf{VA}^+(M) = \mathsf{VA}(M)$.*

Rationally labeled transducers When we eliminate ε-transitions, we take the perspective that a valence transducer with ε-transitions performs big steps consisting of one transition with input and a sequence of ε-transitions. Then, the set of all elements added to the storage and the output during such a big step in a valence transducer over M with output in C is a rational subset of $M \times C$. In other words, we will consider rationally labeled transducers.

We give a formal definition. A *rationally labeled valence transducer over M with output in C* is an automaton over $X^* \times \mathsf{Rat}(M \times C)$. For the automaton $A = (Q, X^* \times \mathsf{Rat}(M \times C), E, q_0, F)$, we also write $A = (Q, X, M, C, E, q_0, F)$. The *transduction performed by* A is

$$\mathsf{T}(A) = \{(w, c) \in X^* \times C \mid \exists q \in F\colon (q_0, (\varepsilon, \{1\})) \to_A^* (q, (w, S)),\ (1, c) \in S\}.$$

We call A *spelling* if $E \subset Q \times X \times \mathsf{Rat}(M \times C) \times Q$, that is, if it reads exactly one letter in each transition.

The definition of $\mathsf{T}(A)$ for rationally labeled valence transducers A means that A behaves as if instead of an edge $(p, (w, S), q)$, $S \in \mathsf{Rat}(M \times C)$, it had an edge (p, w, m, c, q) for each $(m, c) \in S$. Therefore, in slight abuse of terminology, we will also say that

$$q_0 \xrightarrow{(x_1, m_1, c_1)} q_1 \to \cdots \to q_{n-1} \xrightarrow{(x_n, m_n, c_n)} q_n$$

is a computation in A when there are edges $(q_{i-1}, (x_i, S_i), q_i) \in E$ such that $(m_i, c_i) \in S_i$ for $1 \leqslant i \leqslant n$. A similar custom applies for steps.

The following explains why it suffices to consider rationally labeled valence transducers.

Lemma 8.4.3. *For each valence transducer A over M with output in C, there is a spelling rationally labeled valence transducer A' with $\mathsf{T}(A') = \mathsf{T}(A)$.*

Proof. Let $A = (Q, X, M, C, E, q_0, F)$. We obtain the ε-free rationally labeled valence transducer $A' = (Q, X, M, C, E', q_0, F)$ as follows. We introduce one edge $(p, (x, S), q)$ for every triple $(p, x, q) \in Q \times X \times Q$ such that $S \subseteq M \times C$ is the rational set of elements spelled by paths in A that start in p, go along a number of ε-edges, then pass through an edge labeled x and then again go along a number of ε-edges and stop in q. Then clearly $\mathsf{T}(A') = \mathsf{T}(A)$. □

In Sections 8.4.1 to 8.4.3, we will show that every monoid in SC^- is strongly ε-independent.

8.4.1 One partially blind counter

This section is devoted to the proof of the following lemma.

Lemma 8.4.4. $\mathbb{B}$ *is strongly* ε*-independent.*

In order to prove it, we need an auxiliary lemma.

Lemma 8.4.5. *For each valence transducer over* $\mathbb{B}$ *with output in* C*, there is an equivalent rationally labeled valence transducer* $\hat{A} = (Q, X, M, C, E, q_0, F)$ *such that every edge in* $\hat{A}$ *is of the form* $(p, x, R^{\oplus}(m, c)L^{\oplus}, q)$ *with* $p, q \in Q$*,* $R \in \mathsf{Rat}(\{a\}^{\oplus} \times C)$*,* $(m, c) \in \mathbb{B} \times C$*,* $L \in \mathsf{Rat}(\{\bar{a}\}^{\oplus} \times C)$*, and such that in every computation*

$$q_0 \xrightarrow{(x_1, R_1^{\oplus}(m_1, c_1)L_1^{\oplus})} q_1 \cdots q_{n-1} \xrightarrow{(x_n, R_n^{\oplus}(m_n, c_n)L_n^{\oplus})} q_n,$$

we have

$$R_1 \subseteq R_2 \subseteq \cdots \subseteq R_n \text{ and } L_1 \supseteq L_2 \supseteq \cdots \supseteq L_n. \tag{8.5}$$

Proof. Let $T \in \mathsf{VT}(\mathbb{B}, C)$. We may assume that $T = \mathsf{T}(A)$ for a rationally labeled valence transducer $A = (Q, X, \mathbb{B}, C, E, q_0, F)$ over $\mathbb{B}$ with output in C. By Proposition 8.3.1, we may also assume that every edge in A has the form (p, x, LR, q), with $L \in \mathsf{Rat}(\{\bar{a}\}^{\oplus} \times C)$ and $R \in \mathsf{Rat}(\{a\}^{\oplus} \times C)$.

We may further assume that edges starting in the initial state q_0 are of the form (q_0, x, R, p) and, analogously, edges ending in a final state $q \in F$ are of the form (p, x, L, q), $p \in Q$, $L \in \mathsf{Rat}(\{\bar{a}\}^{\oplus} \times C)$ and $R \in \mathsf{Rat}(\{a\}^{\oplus} \times C)$. Thus, we can construct an equivalent transducer $A' = (Q', X, \mathbb{B}, C, E', q_0, F')$ each edge of which simulates the R-part of one edge of A and then the L-part of another edge of A. Hence, in A', every edge is of the form (p, x, RL, q) with $p, q \in Q'$, $R \in \mathsf{Rat}(\{a\}^{\oplus} \times C)$, and $L \in \mathsf{Rat}(\{\bar{a}\}^{\oplus} \times C)$.

Since $\{a\}^{\oplus} \times C$ and $\{\bar{a}\}^{\oplus} \times C$ are commutative, all such R and L are semilinear sets and we can even assume that every edge is of the form

$$(p, x, R^{\oplus}(m, c)L^{\oplus}, q),$$

in which $(m, c) \in \mathbb{B} \times C$ and where $R \subseteq \{a\}^{\oplus} \times C$ and $L \subseteq \{\bar{a}\}^{\oplus} \times C$ are finite subsets.

The crucial observation of this lemma is that if we allow the transducer to apply elements of $\{a\}^{\oplus} \times C$ that, in an edge $(p, x, R^{\oplus}(m, c)L^{\oplus}, q)$ traversed earlier, were contained in R, we do not increase the set of accepted pairs in $X^* \times C$. This is due to the fact that if the counter realized by $\mathbb{B}$ does not go below zero in this new computation, it will certainly not go below zero if we add the value at hand in an earlier step. Thus, any computation in the new transducer can be transformed into one in the old transducer. Furthermore, the commutativity of C guarantees that the output is invariant under this transformation. Analogously, if we allow the transducer to apply elements from $\{\bar{a}\}^{\oplus} \times C$, as long as it ensures that in some edge $(p, x, R^{\oplus}(m, c)L^{\oplus}, q)$ traversed later, they are contained in L, we do not change the accepted set of pairs either.

Therefore, we construct a rationally labeled transducer $\hat{A}$ from A'. In its state, $\hat{A}$ stores a state of A' and two sets: a finite set $\tilde{R} \subseteq \{a\}^{\oplus} \times C$ and a finite set $\tilde{L} \subseteq \{\bar{a}\}^{\oplus} \times C$. $\tilde{R}$ always contains all those elements of $\{a\}^{\oplus} \times C$ that have occurred

in sets R so far, and $\tilde{L}$ are elements of $\{\bar{a}\}^{\oplus} \times C$ that still have to be encountered in sets L in the future. Then for every edge $(p, x, R^{\oplus}(m, c)L^{\oplus}, q)$ in A', we have an edge labeled $(x, (R \cup \tilde{R})^{\oplus}(m, c)(L \cup \tilde{L})^{\oplus})$. $\hat{A}$ will then add the elements of R to its set $\tilde{R}$ and nondeterministically remove some elements of L from $\tilde{L}$ (they can only be removed if this is their last occurrence; otherwise, we might need them in $\tilde{L}$ later). The final state will then make sure that $\tilde{L}$ is empty and $\hat{A}$ has thus only applied elements early that would later appear. In the initial state, both sets $\tilde{R}$ and $\tilde{L}$ are empty and then $\tilde{L}$ is filled nondeterministically. By construction, $\hat{A}$ satisfies Eq. (8.5). □

We are now ready to show that $\mathbb{B}$ is strongly ε-independent.

Proof of Lemma 8.4.4. Let $T \in VT(\mathbb{B}, C)$. By Lemma 8.4.5, we have $T = T(A)$ for some rationally labeled valence transducer over M with output in C with the property stated there. The essential idea of the proof is to simulate a certain fragment of all computations of A, namely those where in every edge the element in the $R^{\oplus}$-part and the element in the $L^{\oplus}$-part differ in size (as measured by the number of a and $\bar{a}$) only by a bounded amount. We will see that every pair in $X^* \times C$ can be produced through such a computation by showing that if in some edge the $R^{\oplus}$-part and the $L^{\oplus}$-part differ by more than the bound, either some part of the $R^{\oplus}$-part can be postponed or some part of the $L^{\oplus}$-part can be applied earlier. Note that the property of Lemma 8.4.5 allows us to postpone elements of $R^{\oplus}$-parts and apply elements of $L^{\oplus}$ earlier.

Simulating the computations in this fragment is then simple: Since for each occurring difference (between the sizes), the set of possible outputs is semilinear, we only have to output the semilinear set and add the difference.

Let us define the new transducer $\hat{A}$ that will simulate the fragment. It is obtained from A as follows. Let $e = (p, x, R^{\oplus}(a^k\bar{a}^n, c)L^{\oplus}, q)$ be an edge in A. Let Y and Z be alphabets in bijection with R and L, respectively, and define $\varphi\colon (Y \cup Z)^{\oplus} \to \mathbb{B} \times C$ to be the morphism extending these bijections. In the following, we write π_1 and π_2 for the projection on the left and right component, respectively. Then, if $\kappa\colon \mathbb{B} \to \mathbb{Z}$ is the morphism with $\kappa(a) = 1$ and $\kappa(\bar{a}) = -1$, let $\psi\colon (Y \cup Z)^{\oplus} \to \mathbb{Z}$ be defined by $\psi(\mu) = \kappa(\pi_1(\varphi(\mu)))$. The set $C_i = \pi_2(\varphi(\psi^{-1}(i))) \subseteq C$ now contains all outputs $c_1c_2 \in C$ such that there are $(a^t, c_1) \in R^{\oplus}$ and $(\bar{a}^u, c_2) \in L^{\oplus}$ with $t - u = i$. Moreover, since $\psi^{-1}(i) \subseteq (Y \cup Z)^{\oplus}$ is clearly Presburger definable, the set C_i is semilinear. Let

$$b = \min\{-1, \psi(z) + n - k \mid z \in Z\},$$
$$B = \max\{1, \psi(y) + n - k \mid y \in Y\}.$$

$\hat{A}$ has the same set of states as A. To simulate the edge e, we introduce for each $i \in \mathbb{N}$ with $b < i < B$ the edge

$$(p, x, a^{k+i}\bar{a}^n, cC_i, q) \qquad \text{if } i \geqslant 0, \tag{8.6}$$
$$(p, x, a^k\bar{a}^{n-i}, cC_i, q) \qquad \text{if } i < 0. \tag{8.7}$$

Initial state and final states remain unaltered. We claim that $\Phi(T(\hat{A})) = T(A)$.

The transducer $\hat{A}$ is constructed so as to simulate all computations in A where the difference between the element in the $R^{\oplus}$-part and the element in the $L^{\oplus}$-part is between b and B. Therefore, we have $\Phi(T(\hat{A})) \subseteq T(A)$.

It remains to be shown that every computation in A has an equivalent computation in our fragment and hence in $\hat{A}$. Here, the idea is that if we transform a computation in A by postponing elements of $R^{\oplus}$-parts and applying elements of $L^{\oplus}$-parts early as much as possible, we end up with a computation in our fragment. Consider a computation in A containing a step

$$p \xrightarrow{(x,r(a^k\bar{a}^n,c)\ell)} q \quad \text{for an edge} \quad (p, x, R^{\oplus}(a^k\bar{a}^n, c)L^{\oplus}, q).$$

Define Y, Z, φ, κ, ψ, b, B as above. Then there are $\mu \in Y^{\oplus}$ and $\nu \in Z^{\oplus}$ with $\varphi(\mu) = r$ and $\varphi(\nu) = \ell$.

Let us identify the situations in which we can postpone elements in μ or apply elements of ν earlier. Suppose there is a y in μ such that

$$\psi(\mu - y) + k - n + \psi(\nu) \geqslant 0, \tag{8.8}$$

that is, the counter stays above zero until the end of the step, even if we do not add y. Then the counter will also stay above zero if we postpone the application of $\varphi(y)$ until the beginning of the next step. By construction, A allows us to do so. Note that we cannot be in the last step of the computation, since this would leave a positive value on the counter. Analogously, suppose there is a z in ν such that

$$-\psi(\nu - z) + n - k - \psi(\mu) \geqslant 0, \tag{8.9}$$

that is, when starting from the right (and interpreting $\bar{a}$ as increment and a as decrement), the counter does not drop below zero until the beginning of the step, even if we do not apply $\varphi(z)$. Then we can apply $\varphi(z)$ earlier in the computation. Again, note that this cannot happen in the first step, since this would mean the computation starts by subtracting from the counter.

We transform the computation in the following way. Whenever in some step, Eq. (8.8) is satisfied, we move $\varphi(y)$ to the right (i.e., we postpone the application of $\varphi(y)$). Symmetrically, whenever in some step, Eq. (8.9) is fulfilled, we move $\varphi(z)$ to the left (i.e., we apply $\varphi(z)$ earlier). We repeat this and since the computation is finite, this process will terminate and we are left with a valid equivalent computation in which Eqs. (8.8) and (8.9) do not occur. We will show that this means we arrived at a computation in our fragment. Note that we are in the fragment if in each step, we have $b < \psi(\mu) + \psi(\nu) < B$.

Equations (8.8) and (8.9) are equivalent to

$$\psi(\mu) + \psi(\nu) \geqslant \psi(y) + n - k,$$
$$\psi(\mu) + \psi(\nu) \leqslant \psi(z) + n - k.$$

Since these are not satisfied for any $y \in Y$ and $z \in Z$, we have

$$\psi(\mu) + \psi(\nu) < \psi(y) + n - k \qquad \text{for each } y \text{ in } \mu, \tag{8.10}$$
$$\psi(\mu) + \psi(\nu) > \psi(z) + n - k \qquad \text{for each } z \text{ in } \nu \tag{8.11}$$

and thus

$$b < \psi(\mu) + \psi(\nu) < B.$$

Indeed, the left inequality follows from Eq. (8.11) if $\nu \neq 0$. If $\nu = 0$, then we have $b < 0 \leqslant \psi(\mu) + \psi(\nu)$. Symmetrically, the right inequality follows from Eq. (8.10) if $\mu \neq 0$. If $\mu = 0$, then $\psi(\mu) + \psi(\nu) \leqslant 0 < B$. This means, the resulting computation is in our fragment and each step has a counterpart in the edges (8.6) and (8.7). Therefore, $\Phi(T(\hat{A})) = T(A)$. □

8.4.2 Blind counters

Our next step is to show that adding a blind counter preserves the property of strong ε-independence. Let us sketch the idea for showing $\mathsf{VA}^+(\mathbb{Z}) = \mathsf{VA}(\mathbb{Z})$. This boils down to simulating the addition of a semilinear subset of $\mathbb{Z}$ by adding single elements of $\mathbb{Z}$. Here, our strategy is to add the constant element of a linear set and then a bounded number of period elements. This means we have to show that if the sum of period elements (added throughout the whole computation) allows to arrive at 0, then a bounded number in each step already suffices. The following lemma provides such a bound.

Lemma 8.4.6. *Let $\varphi\colon X^\oplus \to \mathbb{Z}$ be a morphism. Then for any $n \in \mathbb{Z}$, the set $\varphi^{-1}(n)$ is semilinear. In particular, $\ker\varphi$ is finitely generated. Furthermore, there is a constant $k \in \mathbb{N}$ such that for any $\mu \in X^\oplus$, there is a $\nu \leqslant \mu$ with $\mu \in \nu + \ker\varphi$ and $|\nu| \leqslant k \cdot |\varphi(\mu)|$.*

Proof. Since $\varphi^{-1}(n) \subseteq X^\oplus$ is clearly Presburger definable, it is semilinear. This implies that $\ker\varphi$ is finitely generated: If $\ker\varphi = \bigcup_{i=1}^{n} \mu_i + F_i^\oplus$, then for each $\mu \in F_i$, we have $\varphi(\mu_i) = 0$ and $\varphi(\mu_i + \mu) = 0$ and hence $\varphi(\mu) = 0$. This means $F_i \subseteq \ker\varphi$ and thus $\ker\varphi = \big(\{\mu_1, \dots, \mu_n\} \cup \bigcup_{i=1}^{n} F_i\big)^\oplus$.

In order to prove the second claim, we present an algorithm to obtain ν from μ, from which it will be clear that the size of ν is linear in the absolute value of $\varphi(\mu)$. Without loss of generality, let $\varphi(\mu) > 0$. The algorithm operates in two phases.

In the first phase, we construct a $\nu \leqslant \mu$ with $\varphi(\nu) \geqslant \varphi(\mu) - m$ such that $|\nu|$ is linear in $|\varphi(\mu)|$, where $m = \max\{|\varphi(x)| \mid x \in X\}$. In this phase, we start with $\nu = 0$ and successively add elements from μ to ν until $\varphi(\nu) \geqslant \varphi(\mu) - m$. As long as we still have $\varphi(\nu) < \varphi(\mu) - m$, it is guaranteed that we find an $x \in X$ such that $\nu + x \leqslant \mu$ and $\varphi(x) > 0$. Thus, after at most $\varphi(\mu) - m$ steps, we have $\varphi(\nu) \geqslant \varphi(\mu) - m$ and $|\nu| \leqslant \varphi(\mu) - m$. Since we stopped after we first had $\varphi(\nu) \geqslant \varphi(\mu) - m$, we also have $\varphi(\mu) - m \leqslant \varphi(\nu) \leqslant \varphi(\mu) + m$.

In the second phase, we successively extend ν such that $\varphi(\nu)$ always stays within the interval $[\varphi(\mu) - m, \varphi(\mu) + m]$: If $\varphi(\nu) < \varphi(\mu)$, we can find an $x \in X$ with $\nu + x \leqslant \mu$ and $\varphi(x) > 0$ and if $\varphi(\nu) > \varphi(\mu)$, we can find an $x \in X$ with $\nu + x \leqslant \mu$ and $\varphi(x) < 0$. At some point, we have to arrive at $\varphi(\nu) = \varphi(\mu)$. Our procedure is nondeterministic and we consider a computation with a minimal number of additions to ν. Then, no value $\varphi(\nu)$ occurs more than once: Otherwise, we could have left out the summands between the two occurrences.

Hence, the values $\varphi(\nu)$ obtained in the course of the second phase are pairwise distinct numbers in $[\varphi(\mu) - m, \varphi(\mu) + m]$. Therefore, the second phase adds at most $2m + 1$ elements to ν. This means $|\nu| \leqslant \varphi(\mu) - m + 2m + 1$. □

Lemma 8.4.7. *Suppose $M \in \mathsf{SC}^-$ is strongly ε-independent. Then $M \times \mathbb{Z}$ is strongly ε-independent as well.*

Proof. We denote the operation of C by $+$. Let $T \in \mathsf{VT}(M \times \mathbb{Z}, C)$ and suppose $A = (Q, X, M \times \mathbb{Z}, C, E, q_0, F)$ is a transducer for T. By letting

$$E' = \{(p, x, m, (z, c), q) \mid (p, x, (m, z), c, q) \in E\},$$

we get a valence transducer $A' = (Q, X, M, \mathbb{Z} \times C, E', q_0, F)$ over M with output in $\mathbb{Z} \times C$. Then we have $(w, c) \in T$ if and only if the pair $(w, (0, c))$ is contained in

$T(A')$. Since M is strongly ε-independent, there is an ε-free valence transducer A'' over M with output in $\mathsf{SL}(\mathbb{Z} \times C)$ such that $\Phi(T(A'')) = T(A')$.

In A'', every edge is of the form (p, x, m, S, q), where $S \subseteq \mathbb{Z} \times C$ is semilinear. Thus, we may assume that every edge is of the form $(p, x, m, (\ell, c) + S^{\oplus}, q)$, where $S \subseteq \mathbb{Z} \times C$ is finite. Since $\mathbb{Z} \times C$ is commutative, we do not change the transduction if we output elements $s \in \mathbb{Z} \times C$ that occur in some S in a step anywhere else in the computation. Therefore, we can transform A'' so as to make it guess the set $\tilde{S}$ of all $s \in \mathbb{Z} \times C$ that will occur in an S somewhere in the computation. It uses its finite control to guarantee that the computation is only accepting if all elements of $\tilde{S}$ actually occur. In every step, it allows the application of every element of $\tilde{S}^{\oplus}$. Thus, in the resulting transducer A''', we have that in any computation, the set S in steps $p \xrightarrow{(x,m,(\ell,c)+S^{\oplus})} q$ does not change throughout the computation.

The construction of the new ε-free valence transducer $\hat{A}$ over $M \times \mathbb{Z}$ relies on the following idea. In order to simulate an edge $(p, x, m, (\ell, c) + S^{\oplus}, q)$, we add ℓ to the storage, output c, and output a bounded number of elements of S. In addition, each step outputs the semilinear set of all those elements of $S^{\oplus}$ that leave the $\mathbb{Z}$-component unchanged (whose output hence complies with the behavior of A''').

The transducer $\hat{A}$ has the same set of states as A''' and the edges are defined as follows. For the edge $e = (p, x, m, (\ell, c) + S^{\oplus}, q)$ in A''', let Y be an alphabet in bijection with S and let $\varphi \colon Y^{\oplus} \to \mathbb{Z} \times C$ be the morphism extending this bijection. Moreover, define the maps

$$\begin{aligned} \alpha \colon Y^{\oplus} &\longrightarrow \mathbb{Z}, & \beta \colon Y^{\oplus} &\longrightarrow C, \\ \mu &\longmapsto \pi_1(\varphi(\mu)), & \mu &\longmapsto \pi_2(\varphi(\mu)), \end{aligned}$$

where $\pi_1 \colon \mathbb{Z} \times C \to \mathbb{Z}$ and $\pi_2 \colon \mathbb{Z} \times C \to C$ are the projection onto the left and right component, respectively. Hence, for $\mu \in Y^{\oplus}$, $\alpha(\mu)$ is μ's $\mathbb{Z}$-effect and $\beta(\mu)$ is the output created by μ. Let $k \in \mathbb{N}$ be the constant provided by Lemma 8.4.6 for the map α and let B be the maximum over all values $|\ell'|$ for edges $(p', x', m', (\ell', c') + S^{\oplus}, q')$ in A'''. In lieu of e, we give $\hat{A}$ an edge

$$(p,\ \ x,\ \ (m, \ell + \alpha(\nu)),\ \ c + \beta(\nu + \ker \alpha)),\ \ q)$$

for each $\nu \in Y^{\oplus}$ with $|\nu| \leqslant k \cdot B$. Initial and final states remain unaltered. Note that by Lemma 8.4.6, the set $c + \beta(\nu + \ker \psi) \subseteq C$ is semilinear. Coming back to our description above, this edge adds ℓ to the storage, outputs c, applies the (bounded number of) elements in $\varphi(\nu) \in \mathbb{Z} \times C$, and outputs the semilinear set $\beta(\ker \alpha)$. Observe that each of these edges simulates an step in A''': It chooses an element of $S^{\oplus}$, namely $\varphi(\mu)$ for some $\mu \in \nu + \ker \alpha \subseteq Y^{\oplus}$, adds $\ell + \alpha(\nu) = \ell + \alpha(\mu)$ to the $\mathbb{Z}$-component of the storage and outputs $c + \beta(\mu)$. Therefore, if $(w, c) \in \Phi(T(\hat{A}))$, then $(w, (0, c)) \in \Phi(T(A'''))$ and thus $(w, c) \in T$.

It remains to be shown that $(w, (0, c)) \in \Phi(T(A'''))$ implies $(w, c) \in \Phi(T(\hat{A}))$. Therefore, Let

$$q_0 \xrightarrow{(x_1,m_1,(\ell_1,c_1)+s_1)} q_1 \cdots q_{n-1} \xrightarrow{(x_n,m_n,(\ell_n,c_n)+s_n)} q_n \tag{8.12}$$

be a computation in A''' that witnesses $(w, (0, c)) \in \Phi(T(A'''))$. By construction of A''', there is a finite set S such that $s_i \in S^{\oplus}$ for $1 \leqslant i \leqslant n$. Define Y, φ, α, β, k, B

as above. We claim that we can choose $\nu_i \in Y^{\oplus}$, $|\nu_i| \leqslant kB$, and $\xi_i \in \ker \alpha$ such that the computation with

$$q_{i-1} \xrightarrow{(x_i,\ (m_i,\ \ell_i+\alpha(\nu_i)),\ c_i+\beta(\nu_i+\xi_i))} q_i \tag{8.13}$$

has the same output as (8.12) (of course, it reads the same input $w = x_1 \cdots x_n$). Intuitively, this means we have to decompose the effect $s_1 + \cdots + s_n \in S^{\oplus}$ into n multisets $\nu_1, \ldots, \nu_n \in Y^{\oplus}$ of bounded size and n multisets $\xi_1, \ldots, \xi_n \in Y^{\oplus}$ that reside in $\ker \alpha$, i.e. act neutrally on the counter.

By definition of Y, is a $\mu_i \in Y^{\oplus}$ with $\varphi(\mu_i) = s_i$. Since the computation (8.12) accepts $(w, (0, c))$, we have $\alpha(\mu_1 + \cdots + \mu_n) + \ell_1 + \cdots + \ell_n = 0$ and for $\mu = \mu_1 + \cdots + \mu_n$ we have thus

$$|\alpha(\mu)| = |\alpha(\mu_1 + \cdots + \mu_n)| = |\ell_1 + \cdots + \ell_n| \leqslant n \cdot B.$$

Lemma 8.4.6 now yields a $\nu \leqslant \mu$ with $\mu \in \nu + \ker \alpha$ and $|\nu| \leqslant knB$. This means that we can decompose $\nu = \nu_1 + \cdots + \nu_n$ such that $|\nu_i| \leqslant kB$ for $1 \leqslant i \leqslant n$. Since $\mu \in \nu + \ker \alpha$, we have

$$\ell_1 + \alpha(\nu_1) + \cdots + \ell_n + \alpha(\nu_n) = \ell_1 + \cdots + \ell_n + \alpha(\mu) = 0.$$

This means the computation (8.13) leaves the counter at zero in the end. The relation $\mu \in \nu + \ker \alpha$ also means that $\mu - \nu \in \ker \alpha$. We can therefore simply choose $\xi_1 = \mu - \nu$ and $\xi_i = 0$ for $i > 1$. With these settings, we have $\sum_{i=1}^{n} \beta(\nu_i + \xi_i) = \beta(\mu) = \sum_{i=1}^{n} \beta(\mu_i)$, so that the computation (8.13) has output

$$\sum_{i=1}^{n} c_i + \beta(\nu_i + \xi_i) = \sum_{i=1}^{n} c_i + \beta(\mu_i) = c.$$

Hence, we have $(w, c) \in \Phi(\mathsf{T}(\hat{A}))$. □

8.4.3 Stacks

The following is the last step in proving Theorem 8.1.1.

Lemma 8.4.8. *Suppose $M \in \mathsf{SC}^-$ is strongly ε-independent. Then, $M * \mathbb{B}$ is strongly ε-independent as well.*

Proof. If $M \cong \mathbf{1}$, then $M * \mathbb{B} \cong \mathbb{B}$ and this case has been treated in Lemma 8.4.4. Hence, we have $M \neq \{1\}$.

By Lemma 8.4.3, in order to show $\mathsf{T} \in \Phi(\mathsf{VT}^+(M * \mathbb{B}, \mathsf{SL}(C)))$ for any given $\mathsf{T} \in \mathsf{VT}(M * \mathbb{B}, C)$, we may assume that $\mathsf{T} = \mathsf{T}(A)$ for a rationally labeled valence transducer A over $M * \mathbb{B}$ with output in C. Without loss of generality, the set of edges E in A satisfy $E \subseteq Q \times X \times \mathsf{Rat}((M * \mathbb{B}) \times C) \times Q$.

Since $M \in \mathsf{SC}^-$ and $M \neq \{1\}$, we have $\mathsf{R}_1(M) \neq \{1\}$. Hence, we have $M * \mathbb{B}^{(n)} \hookrightarrow M * \mathbb{B}$ (Lemma 2.6.5), which means an ε-free valence transducer over $M * \mathbb{B}^{(n)}$ can easily be transformed into one over $M * \mathbb{B}$. Hence, it suffices to show

$$\mathsf{T}(A) \in \Phi(\mathsf{VT}(M * \mathbb{B}^{(n)})\mathsf{SL}(C))$$

for some $n \in \mathbb{N}$. Intuitively, this means we construct a valence transducer that has access to a stack of elements of M that are separated by symbols from an arbitrarily large stack alphabet.

By Proposition 8.3.1, we may assume that for every edge $(p, (x, S), q) \in E$, there is an alphabet $X' = \{x, \bar{x}\} \cup Y \cup Z$, a morphism $\varphi\colon X'^* \to (M * \mathbb{B}) \times C$ with $\varphi(x) = a$, $\varphi(\bar{x}) = \bar{a}$, $\varphi(Y) \subseteq M$, $\varphi(Z) \subseteq C$, and rational languages

$$L \subseteq ((Y \cup Z)^* \bar{x})^*, \quad V \subseteq (Y \cup Z)^*, \quad R \subseteq (x(Y \cup Z)^*)^* \tag{8.14}$$

such that $S = \varphi(LVR)$. Indeed, replacing a rational set S by another one S' with $S \cap (J_1(M * \mathbb{B}) \times C) = S' \cap (J_1(M * \mathbb{B}) \times C)$ does not affect the transduction, since elements outside of $J_1(M * \mathbb{B})$ cannot occur in a product that results in 1. Hence, for each of the sets in the union provided by Proposition 8.3.1, we can introduce an edge $(p, (x, \varphi(LVR)), q)$ that satisfies Eq. (8.14). Moreover, it means no loss of generality to assume that the alphabets X', Y, Z are the same for all edges $(p, (x, S), q)$.

Denotation of rational sets In order to be able to denote several appearing rational sets using a pair of states, we construct finite automata

$$\begin{aligned} B_- &= (Q_-, X', E_-, q_0, \emptyset), \\ B_0 &= (Q_0, Y \cup Z, E_0, q_0, \emptyset), \\ B_+ &= (Q_+, X', E_+, q_0, \emptyset) \end{aligned}$$

such that for each edge $(p, (x, \varphi(LVR)), q) \in E$, we find the sets L, V, and R as $L = L_{r,s}(B_-)$ and $V = L_{t,u}(B_0)$ and $R = L_{v,w}(B_+)$ for some states $r, s \in Q_-$, $t, u \in Q_0$, $v, w \in Q_+$. Because of (8.14), we may assume that the edges in B_-, B_0, and B_+ have labels in $X \cup \{\varepsilon\}$ and there are subsets $\tilde{Q}_- \subseteq Q_-$, $\tilde{Q}_+ \subseteq Q_+$ such that

1. in B_-, an edge is labeled $\bar{x}$ if and only if it enters a state in $\tilde{Q}_-$,
2. an edge in B_+ is labeled x if and only if it leaves a state in $\tilde{Q}_+$, and
3. there are no loops on states in $\tilde{Q}_-$, $\tilde{Q}_+$.

For each $r, s \in \tilde{Q}_-$, $t, u \in Q_0$, $v, w \in \tilde{Q}_+$, let

$$L_{r,s} = \varphi(L_{r,s}(B_-)), \quad V_{t,u} = \varphi(L_{t,u}(B_0)), \quad R_{v,w} = \varphi(L_{v,w}(B_+)). \tag{8.15}$$

The conditions on $\tilde{Q}_-$, $\tilde{Q}_+$ guarantee that

$$L_{r,s} \subseteq (M\bar{a})^* \times C, \qquad V_{t,u} \in \mathsf{Rat}(M \times C), \qquad R_{v,w} \subseteq (aM)^* \times C$$

for any $r, s \in \tilde{Q}_-$, $t, u \in Q_0$, and $v, w \in \tilde{Q}_+$. Moreover, every edge in A is of the form $(p, (x, L_{r,s} V_{t,u} R_{v,w}), q)$.

Note that although $L_{r,s}$ and $R_{v,w}$ are rational subsets of $(M * \mathbb{B}) \times C$ and are included in $(M\bar{a})^* \times C$ and $(aM)^* \times C$, they may not be rational subsets of $(M\bar{a})^* \times C$ and $(aM)^* \times C$. For example, observe that $(M\bar{a})^*$ is a rational subset of $M * \mathbb{B}$, but it is not finitely generated and can thus not be a rational subset of itself. We will therefore speak of *rational sets in* $(M\bar{a})^* \times C$ and $(aM)^* \times C$.

Outputs of cancelations The essential idea of the construction is to maintain on the stack a representation of a set of possibly reached configurations. Roughly speaking, we represent a sequence of rational sets in $(aM)^* \times C$ by elements of $M * \mathbb{B}^{(n)}$. In order to simulate the multiplication of a set $L_{r,s} \subseteq (M\bar{a})^* \times C$, we have to output a set of elements of C that appear as output while canceling out elements on the stack with those in $L_{r,s}$. Therefore, we will output sets of the form

$$C_{v,w,r,s} = \{c \in C \mid (1, c) \in R_{v,w} L_{r,s}\}.$$

Each of these sets is a homomorphic image of a language in $\mathsf{VA}(M * \mathbb{B})$ and since $M * \mathbb{B} \in \mathsf{SC}^-$, these languages are semilinear (Theorem 7.1.1 and Proposition 7.1.2). Hence, each $C_{v,w,r,s}$ is semilinear.

Construction of $\hat{D}$ In the course of a computation, we will have to simulate the multiplication of rational subsets of $M \times C$. Their denotation follows the same principal as $L_{r,s}$, $V_{t,u}$, and $R_{v,w}$. Let

$$\tilde{L}_{r,s} = \{\varphi(w) \mid w \in (Y \cup Z)^*,\ w\bar{x} \in L_{r,s}(B_-)\},$$
$$\tilde{R}_{v,w} = \{\varphi(w) \mid w \in (Y \cup Z)^*,\ xw \in L_{v,w}(B_+)\}$$

for $r, s \in \tilde{Q}_-$ and $v, w \in \tilde{Q}_+$. To simulate their multiplication, we use the hypothesis of M being strongly ε-independent in the following way. We consider

$$W = \tilde{Q}_- \times \tilde{Q}_- \ \cup\ Q_0 \times Q_0 \ \cup\ \tilde{Q}_+ \times \tilde{Q}_+$$

as an alphabet. Let $D = (\{q\}, W, M, C, E', q, \{q\})$ be the rationally labeled valence transducer over M with output in C with the following edges:

- for each $r, s \in \tilde{Q}_-$, one loop on q with input $(r, s) \in W$ and label $\tilde{L}_{r,s}$,
- for each $t, u \in Q_0$, one loop on q with input $(t, u) \in W$ and label $V_{t,u}$, and
- for each $v, w \in \tilde{Q}_+$, one loop on q with input $(v, w) \in W$ and label $\tilde{R}_{v,w}$.

Since M is strongly ε-independent, we can transform D into an ε-free valence transducer $\hat{D} = (\hat{Q}, X, M, \mathsf{SL}(C), \hat{E}, q_0, \hat{F})$ over M with output in $\mathsf{SL}(C)$ such that $\Phi(\mathsf{T}(\hat{D})) = \mathsf{T}(D)$.

Encoding of rational sets on the stack As mentioned above, we will encode rational sets in $(aM)^* \times C$ by elements of $M * \mathbb{B}^{(n)}$. The monoid structure of $M * \mathbb{B}^{(n)}$ allows us to use the positive generators of the n instances of $\mathbb{B}$ as stack symbols. In the simplest case, we represent the set $R_{v,w}$ by a symbol[2] $\bigcirc^{v,w}$, which is available for each $v, w \in \tilde{Q}_+$.

Split By construction, composing an element of $R_{v,w}$ with one of $L_{r,s}$ always yields one that, in the $M * \mathbb{B}$-component, agrees with an element of $R_{v,z}$ for some $z \in \tilde{Q}_+$, or resides outside of $J_1(M * \mathbb{B}) \times C$. Therefore, in order to simulate a computation where an element of $L_{r,s}$ cancels out part of an

[2] In this encoding, we deviate from our custom to put state pairs in the subscript (as in $L_{r,s}(\cdot)$), because these new symbols are often followed by commas, which would put the latter at risk of being confused with primes.

element of $R_{v,w}$, we have a *split* operation, which removes a symbol $\bigcirc^{v,w}$ from the top and puts $\bigcirc^{v,z}\bigcirc^{z,w}$ in its place, so that we can later simulate canceling an element of $R_{z,w}$ by an element of $L_{r,s}$.

Merge and cancel In order to simulate an element of $L_{r,s}$ that cancels out an element in the composition of more than one rational set, we need a way to merge two representations of rational sets. However, if we would merge two representations of rational sets in $(aM)^* \times C$ into one, the resulting representation would not be of the form $\bigcirc^{v,w}$: The elements in $R_{v,z}R_{z,w}$ are only those in $R_{v,w}$ where the state z was visited on the way. Hence, such a representation would need to keep track of such intermediate states. Furthermore, the more representations we would merge, the more information we would have to maintain.

Therefore, we will not merge representations of the form $\bigcirc^{v,w}$. Instead, we have another kind of symbols: The symbol $\oslash^{r,s}$ stands for an element of $(aM)^* \times C$ that can be canceled out by one of $L_{r,s}$. Furthermore, the occurrence of such a symbol also implies that the corresponding output of the canceling process has already been performed. This means, the symbol $\oslash^{r,s}$ is produced by an operation *cancel* that removes $\bigcirc^{v,w}$, places $\oslash^{r,s}$ on top and outputs $C_{v,w,r,s}$. Since C is commutative, this early output does not change the result. The *merge* operation then consists of removing $\oslash^{r,t}\oslash^{t,s}$ and putting $\oslash^{r,s}$ on top. Since we will make sure that we can always assume that a symbol $\bigcirc^{v,w}$ has already been turned into a $\oslash^{r,s}$, the simulation of $L_{r,s}$ amounts to a mere deletion of $\oslash^{r,s}$.

Note that this way, keeping track of intermediate states is not necessary: Since we already performed the necessary output, the only information we need is that $\oslash^{r,s}$ represents an element that can be canceled out by one in $L_{r,s}$.

Convert-to and convert-from Finally, we have to simulate the application of sets $V_{t,u}$. To this end, we hand over control to the transducer $\hat{D}$. This, in turn, is done by storing the state information of $\hat{D}$ in symbols $\square^p$ on the stack. Applying $V_{t,u}$ then means simulating $\hat{D}$ as it uses an edge $(p, (t, u), m, S, q)$. Thus, we apply $V_{t,u}$ by removing $\square^p$ from the stack, using S as output, and adding $m\square^q$ on the stack.

In order to let elements of M that are factors of elements in $R_{v,w}$, i.e., elements of $\tilde{R}_{v,w}$, interact with sets $V_{r,s}$, we have two further operations: *convert-to* and *convert-from*. Convert-to-M removes an element $\bigcirc^{v,w}$ from the stack and instead adds $\square m\square^q$ on the stack and outputs S, provided that $(q_0, (v, w), m, S, q)$ is an edge in $\hat{D}$. That is, the element represented by $\bigcirc^{v,w}$ can be thought of as being handed over to $\hat{D}$. Here, $\square$ represents the a that was part of $R_{v,w}$, but not of $\tilde{R}_{v,w}$. Thus, convert-from-M initiates a subsequence of stack elements that simulate a computation of $\hat{D}$. On the other hand, convert-from-M will terminate such a subsequence by simulating the multiplication of a set of the form $\tilde{L}_{r,s}$. It removes $\square^q$, adds m, removes $\square$, adds $\oslash^{r,s}$, and outputs S, where $(q, (r, s), m, S, f)$ is an edge in $\hat{D}$ and f is a final state of $\hat{D}$. We leave $\oslash^{r,s}$ on the stack to signify that the simulation of $\tilde{L}_{r,s}$ has been carried out (before A warranted it), including the corresponding output.

Operations formally Let Θ be the alphabet

$$\Theta = \{\bigcirc^{v,w}, \oslash^{r,s}, \square^{q}, \square \mid v, w \in \tilde{Q}_+,\ r, s \in \tilde{Q}_-,\ q \in \hat{Q}\}$$

and let $n = |\Theta|$. We let each of the symbols $x \in \Theta$, together with its counterpart $\bar{x}$, be the generators of one of the instances of $\mathbb{B}$ in $M * \mathbb{B}^{(n)}$. Sometimes, it is necessary to apply one of the aforementioned operations not on top of the stack, but one symbol below the top. Therefore, for the operations *split*, *merge*, and *cancel*, we have a *deep* variant, which nondeterministically removes some $x \in \Theta$, then performs the original operation and then puts x back on top. In some cases, the deep variant itself has a deep version, which goes down one level further.

Formally, an *operation* is a (finite) set of elements of $(M * \mathbb{B}^{(n)}) \times \mathsf{SL}(C)$. In accordance with the explanation above, we have the following operations:

(i) $\mathsf{split} = \{(\overline{\bigcirc^{v,w}} \bigcirc^{v,v'} \bigcirc^{v',w}, \{1\}) \mid v, v', w \in \tilde{Q}_+\}$

(ii) $\mathsf{deep\text{-}split} = \{(\bar{x}sx, S) \mid x \in \Theta,\ (s, S) \in \mathsf{split}\}$

(iii) $\mathsf{cancel} = \{(\overline{\bigcirc^{v,w}} \oslash^{r,s}, C_{v,w,r,s}) \mid v, w \in \tilde{Q}_+,\ r, s \in \tilde{Q}_-\}$

(iv) $\mathsf{deep\text{-}cancel} = \{(\bar{x}sx, S) \mid x \in \Theta,\ (s, S) \in \mathsf{cancel}\}$

(v) $\mathsf{deep\text{-}deep\text{-}cancel} = \{(\bar{x}sx, S) \mid x \in \Theta,\ (s, S) \in \mathsf{deep\text{-}cancel}\}$

(vi) $\mathsf{merge} = \{(\overline{\oslash^{r,r'} \oslash^{r',s}} \oslash^{r,s}, \{1\}) \mid r, r', s \in \tilde{Q}_-\}$

(vii) $\mathsf{deep\text{-}merge} = \{(\bar{x}sx, S) \mid x \in \Theta,\ (s, S) \in \mathsf{merge}\}$

(viii) $\mathsf{deep\text{-}deep\text{-}merge} = \{(\bar{x}sx, S) \mid x \in \Theta,\ (s, S) \in \mathsf{deep\text{-}merge}\}$

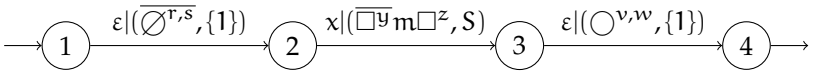
(ix) $\mathsf{convert\text{-}to} = \{(\overline{\bigcirc^{v,w}} \square m \square^{q}, S) \mid v, w \in \tilde{Q}_+,\ (q_0, (v, w), m, S, q) \in \hat{E}\}$

(x) $\mathsf{convert\text{-}from} = \{(\overline{\square^{q}} m \overline{\square} \oslash^{r,s}, S) \mid r, s \in \tilde{Q}_-,\ (q, (r, s), m, S, f) \in \hat{E},\ f \in \hat{F}\}$

(xi) $\mathsf{deep\text{-}convert\text{-}from} = \{(\bar{x}sx, S) \mid x \in \Theta,\ (s, S) \in \mathsf{convert\text{-}from}\}$

Let us describe the transducer $\hat{A}$ in detail. Although $\hat{A}$ has ε-transitions, we will argue later that every element of $T(A)$ is accepted by $\hat{A}$ by a computation with only a bounded number of ε-transitions before and after every non-ε-transition. This clearly permits the removal of ε-transitions from $\hat{A}$.

$\hat{A}$ is obtained from A by first removing all edges and then for each edge $(p, x, L_{r,s} V_{t,u} R_{v,w}, q)$ in A and each edge $(y, (t, u), m, S, z)$ in $\hat{D}$, gluing in the automaton



(8.16)

between p and q. Of course, the three edges from 1 to 4 simulate $L_{r,s}$, $V_{t,u}$, and $R_{v,w}$, respectively. Furthermore, on every state of $\hat{A}$ (including those in the glued in automata), we add loops labeled with the operations (i) to (xi) and reading ε. Finally, we add a loop labeled $(\varepsilon, \oslash^{r,r}, \{1\})$ for each $r \in \tilde{Q}_-$ on the initial state and a loop labeled $(\varepsilon, \overline{\bigcirc^{v,v}}, \{1\})$ for each $v \in \tilde{Q}_+$ on all final states.

It is clear that $\Phi(\mathsf{T}(\hat{A})) \subseteq \mathsf{T}(A)$. It remains to be shown that $\hat{A}$ can accept every pair $(w, c) \in \mathsf{T}(A)$ with an unbounded number of steps before and after each non-ε-transition. Let us begin by describing how (w, c) can be accepted at all. First we bring a symbol $\oslash^{r,r}$ on the stack to represent an empty storage. The first simulated edge $(p, x, L_{r,s} V_{t,u} R_{v,w}, q)$, which has $s = r$, will thus be able to take the edge from state 1 to 2 in (8.16). We assume that on the stack, there are no symbols of the form $\oslash^{r,s}$ except for one representing the empty stack. Thus, the monoid element in the configuration belongs to

$$\{\bigcirc^{v,w},\ \square m \square^{y} \mid v, w \in \tilde{Q}_+,\ m \in M,\ y \in \hat{Q}\}^* \cup \{\oslash^{r,r} \mid r \in \tilde{Q}_-\}.$$

For each edge $(p, x, L_{r,s} V_{t,u} R_{v,w}, q)$ in the computation in A, we execute the following *phases*:

(a) Apply a sequence of cancel/deep cancel, convert-from/deep convert-from, merge, and split/deep-split loops in state 1 to obtain the symbol $\oslash^{r,s}$ on top of the stack. Note that for this, we need to use a split or deep split loop at most once, namely for the lowest used occurrence of a $\bigcirc^{v',w'}$, which might be canceled only partially.

(b) Use an edge $(\varepsilon, \overline{\oslash^{r,s}}, \{1\})$ in (8.16).

(c) If necessary, use a split loop in state 2.

(d) If necessary, use a convert-to loop in state 2.

(e) Choose an edge $(x, \overline{\square^{y}} m \square^{z}, S)$ in (8.16).

(f) Use an edge $(\varepsilon, \bigcirc^{v,w}, \{1\})$ in (8.16).

By 'necessary' in phases (c) and (d), we mean that phase (e) might require that a symbol $\bigcirc^{v,w}$ be converted and perhaps split beforehand.

Note that phase (a) is the only phase that might use an unbounded number of operations. Therefore, we move these operations to an earlier moment (so that they are distributed more equally among the non-ε-transitions). Thereby, we guarantee that before the application of $(\varepsilon, \overline{\oslash^{r,s}}, \{1\})$ (and during the overall simulation of an edge from A) we only need a bounded number of operations. This is done as follows. In each of the phases (a) to (f):

(I) After each introduction of a $\bigcirc^{v,w}$ (by a split/deep split or by adding $\bigcirc^{v,w}$ directly): If this occurrence is eventually canceled after the current phase (without being split), cancel it now (and drop the later cancel). For this, we can use a cancel, deep cancel, or deep deep cancel loop[3].

(II) After each application of a $\overline{\square^{y}} m \square^{z}$: If the resulting subsequence $\square m' \square^{z}$ is eventually converted (without adding another $\overline{\square^{z}} m'' \square^{z'}$), convert it now (and drop the later convert). This can be done using a convert-from loop.

(III) Whenever a symbol $\oslash^{r,s}$ produced by (I) or (II) is eventually merged with the symbol underneath, merge them now (and drop the later merge). This can be done using a merge, deep merge, or deep deep merge loop.

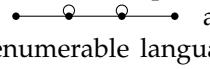
[3]Deep cancel and deep deep cancel are used to cancel the lower result of a split and a deep split, respectively.

Then, in phase (a), any $\bigcirc^{v,w}$ or $\square m \square^z$-subsequence that was canceled or converted and then merged with a symbol underneath in the old computation, is now already merged. Therefore, phase (a) has to do only a bounded number of operations to obtain $\oslash^{r,s}$ on top of the stack: It might still need to (deep) split (the lowest symbol of those turned into $\oslash^{r,s}$ in the old computation, which is only partially canceled), cancel and (deep) merge the resulting symbols, but nothing else. Note that the number of operations introduced by rules (I) through (III) into phase (a) is bounded: In phase (a), only the split or deep split calls for additional operations, but phase (a) executes split or deep split at most once in the old computation.

In the end, the stack contains a symbol $\bigcirc^{v,v}$ for some $v \in \tilde{Q}_+$ to represent the empty storage. This can then be removed by the loop labeled $(\varepsilon, \overline{\bigcirc^{v,v}}, \{1\})$ on the final state.

Thus, any $(w, c) \in T(A)$ can be produced by a computation in $\hat{A}$ using only a bounded number of ε-transitions before and after any input symbol. Hence, $\hat{A}$ can be easily transformed into an equivalent valence transducer with no ε-transitions. □

The foregoing lemmas establish now establish Theorem 8.1.1.

Proof of Theorem 8.1.1. According to Proposition 8.2.4, we have the implications "1 ⇒ 2" and "1 ⇒ 3". By Theorem 4.3.1, if Γ contains  as an induced subgraph, $\mathsf{VA}(\mathbb{M}\Gamma)$ is the class of recursively enumerable languages. This proves "4 ⇒ 5", "2 ⇒ 5", and "3 ⇒ 5". Moreover, by Lemma 8.4.1, if  does not occur, we have $\mathbb{M}\Gamma \in \mathsf{SC}^-$. Hence, each of the conditions 1 to 5 implies condition 6.

Finally, Lemmas 8.4.4, 8.4.7, and 8.4.8 mean that every monoid in SC^- is strongly ε-independent and hence that we have the implication "6 ⇒ 1". □

8.5 Blind and partially blind counters

In this section, we prove Theorem 8.1.3. By Theorem 8.1.1, we already know that when $r \leqslant 1$, we have $\mathsf{VA}^+(\mathbb{B}^r \times \mathbb{Z}^s) = \mathsf{VA}(\mathbb{B}^r \times \mathbb{Z}^s)$. Hence, we only have to show that $\mathsf{VA}^+(\mathbb{B}^r \times \mathbb{Z}^s) \subsetneq \mathsf{VA}(\mathbb{B}^r \times \mathbb{Z}^s)$ if $r \geqslant 2$.

Recall the language L_{bin} from Section 7.2 (Definition 7.2.1):

$$L_{bin} = \{wc^n \mid w \in \{0,1\}^*,\ n \leqslant \mathrm{bin}(w)\}.$$

Greibach1978 [Greibach1978] and, independently, **Jantzen1979 [Jantzen1979]** have shown that L_{bin} can be accepted by a partially blind counter automaton with two counters, but not without ε-transitions. Since we have to show $\mathsf{VA}^+(\mathbb{B}^r \times \mathbb{Z}^s) \subsetneq \mathsf{VA}(\mathbb{B}^r \times \mathbb{Z}^s)$ and we know $L_{bin} \in \mathsf{VA}(\mathbb{B}^r \times \mathbb{Z}^s)$, it suffices to prove $L_{bin} \notin \mathsf{VA}^+(\mathbb{B}^r \times \mathbb{Z}^s)$. We do this by transforming Greibach's and Jantzen's proof into a general property of languages in $\mathsf{VA}(\mathbb{B}^r \times \mathbb{Z}^s)$ (Lemma 8.5.2). We will then apply this to show that $L_{bin} \notin \mathsf{VA}^+(\mathbb{B}^r \times \mathbb{Z}^s)$.

Growth functions and fooling sets Let M be a monoid and $S \subseteq M$ a finite subset. The *growth function* for M and S is defined as $g_{M,S}(n) = |S^{\leqslant n}|$ for $n \in \mathbb{N}$.

From the perspective of valence automata, $g_{M,S}(n)$ is the number of storage contents that a valence automaton can produce after at most n steps if its edges are labeled by S.

It should be mentioned that if S ranges over the generating sets of M (i.e. those for which $M = \langle S \rangle$), then whether $g_{M,S}$ has a polynomial upper bound (or an exponential lower bound) does not depend on the choice of S. It therefore makes sense to speak of finitely generated monoids with *polynomial growth* or *exponential growth*. These notions have been subject to intensive study in group and semigroup theory **[GrigorchukDeLaHarpe1997]**.

Let $n \in \mathbb{N}$ and $L \subseteq X^*$. An n-*fooling set for* $L \subseteq X^*$ is a set $F \subseteq X^{\leqslant n} \times X^*$ such that the following holds:

1. for each $(u, v) \in F$, we have $uv \in L$, and
2. for pairs $(u_1, v_1), (u_2, v_2) \in F$ with $(u_1, v_1) \neq (u_2, v_2)$, we have $u_1 v_2 \notin L$ or $u_2 v_1 \notin L$.

We define the function $f_L : \mathbb{N} \to \mathbb{N}$ as

$$f_L(n) = \max\{|F| \mid F \text{ is an } n\text{-fooling set for } L\}.$$

The notion of fooling sets is often used to prove lower bounds for the number of states in nondeterministic finite automata **[GlaisterShallit1996]**. We use it here to prove lower bounds for growth functions. Our first lemma allows us to derive a lower bound for $g_{M,S}$ under the assumption that L is accepted by an ε-free valence automaton.

Lemma 8.5.1. *Let M be a monoid and $L \in \mathsf{VA}^+(M)$. Then, there is a constant $k \in \mathbb{N}$ and a finite set $S \subseteq M$ such that $f_L(n) \leqslant k \cdot g_{M,S}(n)$ for all $n \in \mathbb{N}$.*

Proof. Let k be the number of states in the automaton for L and let S consist of the elements appearing on edges. Suppose $f_L(n) > k \cdot g_{M,S}(n)$ for some n and let $F = \{(u_1, v_1), \ldots, (u_m, v_m)\}$ be an n-fooling set for L, with $m > k \cdot g_{M,S}(n)$. Since $u_i v_i \in L$ for $1 \leqslant i \leqslant m$, we have an accepting computation for each of these words. Let (q_i, x_i) be the configuration reached in one of these computations after reading u_i, $1 \leqslant i \leqslant m$. Since the automaton has no ε-transitions and has thus passed at most n edges, we have $x_i \in S^{\leqslant n}$ for every $1 \leqslant i \leqslant m$. Furthermore, since $m > k \cdot g_{M,S}(n)$, there are indices $i \neq j$ with $q_i = q_j$ and $x_i = x_j$. This means however, that $u_i v_j \in L$ and $u_j v_i \in L$, contradicting the fooling set condition. □

Together with Lemma 8.5.1, the polynomial growth of $\mathbb{B}^r \times \mathbb{Z}^s$ yields the property of languages in $\mathsf{VA}^+(\mathbb{B}^r \times \mathbb{Z}^s)$.

Lemma 8.5.2. *Let $M = \mathbb{B}^r \times \mathbb{Z}^s$ for $r, s \in \mathbb{N}$ and $S \subseteq M$ a finite set. Then, $g_{M,S}$ is bounded by a polynomial. In particular, for each $L \in \mathsf{VA}^+(\mathbb{B}^r \times \mathbb{Z}^s)$, f_L is bounded by a polynomial.*

Proof. We only prove the first statement, since by Lemma 8.5.1, it implies the second. Every $x \in \mathbb{B}$ can be written uniquely as $x = \bar{a}^k a^\ell$. We define $|x| = k + \ell$. For $y \in \mathbb{Z}$, we have the usual absolute value $|y|$. Thus, for $z \in \mathbb{B}^r \times \mathbb{Z}^s$ and $z = (x_1, \ldots, x_r, y_1 \ldots, y_s)$ we can define

$$|z| = \max\{|x_i|, |y_j| \mid 1 \leqslant i \leqslant r,\ 1 \leqslant j \leqslant s\}.$$

Let $m = \max\{|x| \mid x \in S\}$. Then, for $z \in S^{\leqslant n}$, we have $|z| \leqslant m \cdot n$. There are $k+1$ elements $x \in \mathbb{B}$ with $|x| = k$, meaning that there are

$$1 + \cdots + (k+2) = (k+3) \cdot (k+2)/2$$

elements $x \in \mathbb{B}$ with $|x| \leqslant k$. Furthermore, there are $2k+1$ numbers $y \in \mathbb{Z}$ with $|y| \leqslant k$. Therefore, we have

$$g_{M,S}(n) \leqslant ((m \cdot n + 3)(m \cdot n + 2)/2)^r \cdot (2 \cdot m \cdot n + 1)^s,$$

which is a polynomial in n. □

The following means that if $L = L_{\text{bin}}$, f_L is not bounded by a polynomial and hence $L_{\text{bin}} \notin \mathsf{VA}^+(\mathbb{B}^r \times \mathbb{Z}^s)$ by Lemma 8.5.2.

Lemma 8.5.3. *For $L = L_{\text{bin}}$, we have $f_L(n) \geqslant 2^n$ for every $n \in \mathbb{N}$.*

Proof. Let $n \in \mathbb{N}$, and let F be the set of all pairs (u, v) such that $u \in \{0,1\}^n$ and $v = c^{\text{bin}(u)}$. Then, F is an n-fooling set for L with $|F| \geqslant 2^n$: We have $uv \in L$ for any $(u, v) \in F$. Furthermore, for distinct pairs $(u, v) \neq (u', v')$, we have $u \neq u'$ and we may assume $\text{bin}(u) < \text{bin}(u')$. Then $uv' \notin L$. □

The foregoing lemmas imply Theorem 8.1.3 immediately.

Proof of Theorem 8.1.3. If $r \leqslant 1$, Theorem 8.1.1 already establishes the equation $\mathsf{VA}^+(\mathbb{B}^r \times \mathbb{Z}^s) = \mathsf{VA}(\mathbb{B}^r \times \mathbb{Z}^s)$. If $r \geqslant 2$, we have $L_{\text{bin}} \in \mathsf{VA}(\mathbb{B}^r \times \mathbb{Z}^s)$, but Lemmas 8.5.2 and 8.5.3 together imply that $L_{\text{bin}} \notin \mathsf{VA}^+(\mathbb{B}^r \times \mathbb{Z}^s)$. □

8.6 Conclusion

We have shown that ε-transitions can be avoided in stacked counter automata and, slightly stronger, within each monoid $M \in \mathsf{SC}^-$. This implies NP- and linear time algorithms for the membership problem of each language in F, the class containing all languages of graph-defined storage mechanisms that guarantee semilinearity. Furthermore, we have completely described those combinations of a number of partially blind counters and a number of blind counters for which ε-transitions are avoidable.

The results of this chapter have appeared in [**Zetzsche2013a**].

Related work As mentioned above, the fact that ε-transition can be eliminated for $\mathbb{Z}^n$ and for $\mathbb{B}^{(2)} \times \mathbb{Z}^n$ has also been shown by Latteux [**Latteux1979**] and Hoogeboom [**Hoogeboom2002**], respectively. These are special cases of Theorem 8.1.1, but were unknown to the author at the time of publishing [**Zetzsche2013a**]. It should be noted that the proof of Theorem 8.1.1 is by no means a mere iteration of Hoogeboom's construction. The result here requires showing semilinearity of all involved language classes, adapting the technique of Benois in Proposition 8.3.1 for higher types of storage mechanisms, and preservation of strong ε-independence by building stacks (Lemma 8.4.8), which is the most involved ingredient. The latter has no counterpart in Hoogeboom's proof, which relies on the Greibach normal form of context-free grammars. Finally, Theorem 8.1.1 also shows ε-elimination for monoids $\mathbb{B} \times \mathbb{Z}^n$.

As was also mentioned before, Theorem 8.1.1 also generalizes Greibach's result that ε-transitions can be avoided in pushdown automata.

Directions for future research The first question one might ask is whether this result extends to all graph monoids, say, where the emptiness problem is decidable (otherwise, we cannot expect an effective procedure). However, at this point, this question seems to be of limited interest: Stacked counter automata are already expressively complete for storages with semilinearity (SL). Hence, ε-elimination is available since we can always do this in an equivalent stacked counter automaton.

On the other hand, with those storage mechanisms outside of SL, we can accept the languages in $\mathsf{VA}(\mathbb{B} \times \mathbb{B})$ or even more (Theorem 7.1.1). It seems unlikely that ε-transitions can be eliminated in such powerful models since this would imply NP-algorithms for all their languages. It therefore seems prudent to follow one of the following directions.

1. The most pressing task seems to be to try to *simplify the proof of Theorem 8.1.1*, especially Lemma 8.4.8. A possible approach is to develop a Greibach normal form **[Greibach1965]** for F_i-grammars. However, given the involved nature of the grammar constructions in Section 9.2, it is not clear whether this really simplifies things. A simplification would be of interest because it might lead to new complexity bounds, which brings us to the second direction.

2. Can the elimination be made *more efficient*? This would have direct consequences on the complexity of the rational subset membership problem for graph groups as studied by **LohreySteinberg2008 [LohreySteinberg2008]** (see also Theorem 4.3.9). Suppose a graph group with a decidable rational subset problem is given. Via Theorem 4.3.9 and Proposition 7.1.2, it is not hard to see that for each rational subset, one can construct a stacked counter automaton to whose membership problem one can then reduce the rational subset membership problem. Hence, Theorem 8.1.1 implies that the rational subset membership problem for each fixed subset belongs to NP. An efficient elimination of ε-transitions would therefore entail uniform bounds on this complexity.

Acknowledgements I would like to thank Markus Lohrey for discussions on graph groups.

Chapter 9

Computing downward closures

9.1 Introduction

As mentioned in Section 2.8, Higman [**Higman1952**] in 1952 and Haines [**Haines1969**] in 1969 discovered independently that the subword ordering is a well-quasi-ordering. While regular languages had not been invented at the time of Higman's work, Haines presented the regularity of upward and downward closed sets as the chief application. He also posed as an open problem which languages admit an effective computation of these closures.

In the case of the upward closure, this question was settled for a large range of language classes by **vanLeeuwen1978** [**vanLeeuwen1978**], who proved that when intersection with regular languages is available, upward closures can be computed if and only if emptiness is decidable.

Applications of downward closures The more difficult task, the computation of downward closures, has attracted attention in recent decades. This is due to the fact that downward closures appear to be a promising abstraction. By an abstraction, we mean a simpler object that reflects certain aspects of the abstracted language. Suppose a formal language describes the possible action sequences of a system that is observed through a lossy channel. This means, on the way to the observer, arbitrary actions can get lost. Then, $L{\downarrow}$ is the set of words received by the observer [**HabermehlMeyerWimmel2010**]. Hence, given the downward closure as a finite automaton, we can decide whether two systems are equivalent under such observations, and even whether the behavior of one system includes the other. This is in contrast to the fact that behavioral inclusion for system models themselves is almost always undecidable.

Furthermore, it is well-known that the set of reachable channel contents in lossy channel systems is always downward closed and thus regular. While these reachability sets are not computable in general [**Mayr2003**], methods for computing downward closures can be used to compute reachability sets at least for subclasses of lossy channel systems [**AbdullaBoassonBouajjani2001**].

In addition, compared to other abstractions such as the Parikh image, the downward closure has the advantage of simplifying every language: Most ap-

plications of Parikh images require the semilinearity of the image, which is, of course, not the case for every language. See [**LongCalinMajumdarMeyer2012**, **BachmeierLuttenbergerSchlund2015**] for more applications of downward closures.

Computing downward closures However, while there always *exists* a finite automaton for the downward closure, it seems difficult to *compute* them. In fact, there are few types of systems for which computability has been established and in some cases, they are not computable.

Let us quickly survey the available results on downward closures. Early results are the computability for algebraic extensions and, in particular, context-free languages. The former was shown by **vanLeeuwen1978** [**vanLeeuwen1978**] (see also Theorem 9.1.2) and the latter, using a different approach again by **Courcelle1991** [**Courcelle19** Furthermore, they have been shown to be computable for 0L-systems and context-free FIFO rewriting systems by **AbdullaBoassonBouajjani2001** [**AbdullaBoassonBouajjani2001**]. Finally, **HabermehlMeyerWimmel2010** [**HabermehlMeyerWimmel2010**] have obtained a method for computing downward closures of Petri net languages.

By a reduction of the finiteness problem for Church-Rosser languages, **GruberHolzerKutrib2007** [**G** have shown that these languages do not permit the computation of downward closures. Moreover, using a reduction of the boundedness problem for lossy counter machines, **Mayr2003** [**Mayr2003**] could show that reachability sets of lossy channel systems (which are downward closed) cannot be computed.

We show here that downward closures are computable for the languages in F and thus for all languages accepted by storage mechanisms of graphs monoids that guarantee semilinearity. Equivalently, this means downward closures are computable for stacked counter automata.

Theorem 9.1.1. *Given a language* L *in* F*, one can compute a finite automaton for* $L{\downarrow}$.

One might wonder whether this result is already subsumed by the computability for the algebraic extension of the Petri net languages. After all, these languages also allow combining pushdown storages and counters. This is not the case, since the languages in F allow using counters *parallel* to the pushdown. The algebraic extension of the Petri net languages, however, coincides with the union $\bigcup_{n\geqslant 0} \mathsf{VA}(\mathbb{B} * \mathbb{B}^n)$ (see Theorem 2.6.6), meaning that they are accepted with a stack whose entries are configurations of a Petri net.

In fact, if $L \subseteq X^*$ is a context-free language that is not a Petri net language, such as $(D_1'\#)^*$ [**Jantzen1979**], and $a, b, c \notin X$, it will follow from Proposition 10.2.3 that $K = L \sqcup\!\sqcup \{a^n b^n c^n \mid n \geqslant 0\}$ does not belong to the algebraic extension of the Petri net languages. However, it is easy to see that $K \in \mathsf{F}_1$. On the other hand, since F contains only semilinear languages, it is incomparable to the algebraic extension of the Petri net languages.

Representation of downward closures Let us comment on the representation of downward closures. Since F is an effective full semi-AFL, the downward closure of each language in F is effectively in F. Of course, since downward closed languages are always regular, by "compute the downward closure" we mean "compute a finite automaton for the downward closure". Since the class of downward closed languages is a proper subclass of the regular languages, other descriptional means of have been tailored to them:

- *Ideal decompositions*. An *ideal (over X)* is a language of the form

$$Y_0^*\{x_1, \varepsilon\}Y_1^* \cdots \{x_n, \varepsilon\}Y_n^*,$$

  where the $Y_i \subseteq X$ are subalphabets and the $x_i \in X$ are letters. As shown by **Jullien1969** [**Jullien1969**], downward closed languages are precisely those that can be written as a finite union of ideals. This type of representation was later rediscovered by **Abdulla2004** [**Abdulla2004**], who called it *simple regular expressions*.

- *Obstruction sets*. If $L \subseteq X^*$ is a downward closed language, then the set $X^* \setminus L$ is upward closed and can thus be written as $X^* \setminus L = F{\uparrow}$ for a finite set $F \subseteq X^*$. The finite set can thus be regarded as a representation of $L = X^* \setminus (F{\uparrow})$. Since a word is in L if and only if no word in F appears as a subword, F is called the *obstruction set* for L. These were used, for example, by **Courcelle1991** [**Courcelle1991**] to denote downward closures of context-free languages.

It is not hard to see that all three ways to denote a downward closed language (finite automata, simple regular expressions, and obstruction sets) can be effectively translated into one another [**Abdulla2004**] and it is not necessary to distinguish which representation can be computed.

In our procedure, we will use the following result by van Leeuwen.

Theorem 9.1.2 (van Leeuwen [**vanLeeuwen1978**])**.** *The downward closure is computable for languages in* $\mathsf{Alg}(\mathcal{C})$ *if and only if this is the case for languages in* $\mathcal{C}$*.*

It implies that if there is an algorithm to compute downward closures for F_i, then there is such an algorithm for $G_i = \mathsf{Alg}(F_i)$. However, we are interested in a uniform algorithm for all of F. Fortunately, the algorithm in van Leeuwen's proof is sufficiently uniform to be used in a procedure that works recursively with respect to the levels of the hierarchy $F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \cdots$. Specifically, van Leeuwen's algorithm works by replacing in a $\mathcal{C}$-grammar G each right-hand-side by its downward closure. This yields a Reg-grammar G' with $L(G'){\downarrow} = L(G){\downarrow}$, to which one can then apply the (fixed) algorithm to compute downward closures of context-free languages.

This allows us to compute $L(G){\downarrow}$ for an F_i-grammar G provided that we can compute downward closures for F_i. Hence, it remains the task to compute downward closures of languages $h(L \cap \Psi^{-1}(S))$ for $L \in G_i$. To this end, we will show that we can construct a language in G_i that is larger than $h(L \cap \Psi^{-1}(S))$, but has the same downward closure. This will be accomplished in the following approximation lemma.

Lemma 9.1.3 (Approximation lemma)**.** *Given* $i \in \mathbb{N}$*, a language* $L \subseteq X^*$ *in* G_i*, and a semilinear* $S \subseteq X^{\oplus}$*, one can compute a language* $L' \in G_i$ *that satisfies the inclusions* $L \cap \Psi^{-1}(S) \subseteq L' \subseteq (L \cap \Psi^{-1}(S)){\downarrow}$*.*

The proof of Lemma 9.1.3 relies on the new technique of *Parikh annotations*. It is introduced in the next section and the approximation lemma is proved after the statement of Theorem 9.2.5, our main result on this new concept. First, let us show how downward closures can be computed for F once Lemma 9.1.3 is available.

Proof of Theorem 9.1.1. We perform the computation recursively with respect to the level of the hierarchy $F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \cdots$.

- If $L \in F_0$, then L is finite and we can clearly compute $L{\downarrow}$.
- If $L \in F_i$ with $i \geqslant 1$, then $L = h(L' \cap \Psi^{-1}(S))$ for some $L' \subseteq X^*$ in G_{i-1}, a semilinear $S \subseteq X^{\oplus}$, and a morphism h. Since $h(M){\downarrow} = h(M{\downarrow}){\downarrow}$ for any $M \subseteq X^*$, it suffices to describe how to compute $(L' \cap \Psi^{-1}(S)){\downarrow}$. Using Lemma 9.1.3, we construct a language $L'' \in G_{i-1}$ that satisfies

$$L' \cap \Psi^{-1}(S) \subseteq L'' \subseteq (L' \cap \Psi^{-1}(S)){\downarrow}.$$

  Note that this implies $L''{\downarrow} = (L' \cap \Psi^{-1}(S)){\downarrow}$. This means, the computation of $(L' \cap \Psi^{-1}(S)){\downarrow}$ amounts to recursively computing $L''{\downarrow}$.
- If $L \in G_i$, then L is given by an F_i-grammar G. Using recursion, we compute the downward closure of each right-hand-side of G. We obtain a new Reg-grammar G' by replacing each right-hand-side in G with its downward closure. Then $L(G'){\downarrow} = L{\downarrow}$. Since we can construct a context-free grammar for $L(G')$, we can compute $L(G'){\downarrow}$ using Theorem 9.1.2.

□

Upward closures vs. downward closures One might wonder why downward closures are more difficult to compute than upward closures, given their similar definition. The algorithm for computing the upward closure of a language L is a classical saturation procedure (much like determining the set of productive nonterminals in Section 2.6): We start with an empty set $U_0 = \emptyset$ and, as long as U_i is strictly contained in $L{\uparrow}$, we set $U_{i+1} = (U_i \cup \{w\}){\uparrow}$ for some $w \in L{\uparrow} \setminus U_i$. Since the subword ordering is a well-quasi-ordering, the chain $U_0, U_1, \ldots$ of upward closed sets must become stationary, causing the algorithm to terminate. Here, we can check whether $L{\uparrow} \setminus U_i \neq \emptyset$ by simply intersecting L with a suitable regular language and checking for emptiness.

Why does this approach fail for downward closures? After all, we could try the dual algorithm which starts with X^* as a candidate D_0 for the downward closure and then makes the sets D_i smaller and smaller, this time relying on the fact that descending chains of downward closed sets become stationary. The reason is that we would have to check whether $D_i \subseteq L{\downarrow}$, but checking whether a regular language is contained in a given language is usually undecidable. Note also that this is not just the wrong approach, since computing a finite automaton for $L{\downarrow}$ always enables us to decide whether $R \subseteq L{\downarrow}$ for regular languages R.

This means, the difference has its root in the fact that for 'complex' languages L and regular languages R, it is easy to decide whether $L \subseteq R$, but hard to decide whether $R \subseteq L$. The reason for this, in turn, is that the languages we are interested in are usually generated by nondeterministic processes and thus, a witness for a positive answer to the latter problem would be an infinite set of derivations. Note that in deterministic systems, due to complementation, the two types of decision problems usually differ only in complexity. Simply put, *nondeterminism makes downward closures hard to compute*.

The results in this chapter have appeared in [**Zetzsche2015a**].

9.2 Constructing Parikh annotations

This section introduces Parikh annotations, the key tool in our procedure for computing downward closures. Suppose L is a semilinear language. Then for each $w \in L$, $\Psi(w)$ can be decomposed into a constant vector and a linear combination of period vectors from the semilinear representation of $\Psi(L)$. We call such a decomposition a *Parikh decomposition*. The main purpose of Parikh annotations is to provide transformations of languages that *make reference to Parikh decompositions* without leaving the respective language class. For example, suppose we want to transform a context-free language L into the language L' of all those words $w \in L$ whose Parikh decomposition does not contain a specified period vector. This may not be possible with rational transductions: If $L_\vee = \{a^n b^m \mid m = n \text{ or } m = 2n\}$, then the Parikh image is $(a+b)^\oplus \cup (a+2b)^\oplus$, but a finite state transducer cannot determine whether the input word has a Parikh image in $(a+b)^\oplus$ or in $(a+2b)^\oplus$. Therefore, a Parikh annotation for L is a language K in the same class with additional symbols that allow a finite state transducer (that is applied to K) to access the Parikh decomposition.

Definition 9.2.1. *Let $L \subseteq X^*$ be a language and $\mathcal{C}$ be a language class. A* Parikh annotation (PA) for L in $\mathcal{C}$ *is a tuple $(K, C, P, (P_c)_{c\in C}, \varphi)$, where*

- *C, P are alphabets such that X, C, P are pairwise disjoint,*
- *$K \subseteq C(X \cup P)^*$ is in $\mathcal{C}$,*
- *φ is a morphism $\varphi\colon (C \cup P)^\oplus \to X^\oplus$,*
- *P_c is a subset $P_c \subseteq P$ for each $c \in C$,*

such that

1. *$\pi_X(K) = L$ (the* projection property*),*
2. *$\varphi(\pi_{C\cup P}(w)) = \Psi(\pi_X(w))$ for each $w \in K$ (the* counting property*), and*
3. *$\Psi(\pi_{C\cup P}(K)) = \bigcup_{c\in C} c + P_c^\oplus$ (the* commutative projection property*).*

If $|C| = 1$, then the PA is called linear*. In this case, we also write (K, c, P_c, φ) for the Parikh annotation, where $C = \{c\}$. We say that* Parikh annotations for a language class $\mathcal{C}$ can be constructed *if there is an algorithm that, given a language $L \in \mathcal{C}$, can construct a PA for L in $\mathcal{C}$.*

Intuitively, a Parikh annotation describes for each w in L one or more Parikh decompositions of $\Psi(w)$. The symbols in C represent constant vectors and symbols in P represent period vectors. Here, the symbols in $P_c \subseteq P$ correspond to those that can be added to the constant vector corresponding to $c \in C$. Furthermore, for each $x \in C \cup P$, $\varphi(x)$ is the vector represented by x.

The projection property states that removing the symbols in $C \cup P$ from words in K yields L. The commutative projection property requires that after $c \in C$ only symbols representing periods in P_c are allowed and that all their combinations occur. Finally, the counting property says that the additional symbols in $C \cup P$ indeed describe a Parikh decomposition of $\Psi(\pi_X(w))$.

Clearly, the conditions of a Parikh annotation imply

$$\Psi(L) = \Psi(\pi_X(K)) = \varphi(\pi_{C\cup P}(K)) = \bigcup_{c\in C} \varphi(c) + \varphi(P_c)^{\oplus}$$

and hence that L is semilinear.

Example 9.2.2. *Let* $X = \{a, b, c, d\}$ *and* $L = (ab)^*(ca^* \cup db^*)$. *Then, for*

$$K = e(pab)^*c(qa)^* \cup f(rab)^*d(sb)^*,$$

$P = \{p, q, r, s\}$, *and* $\varphi\colon (C \cup P)^{\oplus} \to X^{\oplus}$ *with*

$$\begin{array}{lll} C = \{e, f\} & \varphi(e) = c, & \varphi(f) = d, \\ P_e = \{p, q\}, & \varphi(p) = a + b, & \varphi(q) = a, \\ P_f = \{r, s\}, & \varphi(r) = a + b, & \varphi(s) = b, \end{array}$$

the tuple $(K, C, P, (P_g)_{g\in C}, \varphi)$ *is a Parikh annotation for* L.

The main purpose of Parikh annotations is to help understand the structure of languages of the form $L \cap \Psi^{-1}(S)$, where $L \subseteq X^*$ and $S \subseteq X^{\oplus}$. Specifically, they provide sufficient conditions for when $L \cap \Psi^{-1}(S)$ is in the same language class as L. Of course, $L \cap \Psi^{-1}(S)$ can be more complex than L: Take, for example, the regular language $L = \{a, b\}^*$ and the semilinear set $S = (a + b)^{\oplus}$. Then $L \cap \Psi^{-1}(S) = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$ is not regular. Now suppose L and $(K, C, P, (P_g)_{g\in C}, \varphi)$ are as in Example 9.2.2. Moreover, S is still the set of multisets with the same number of a's as b's, but now over X. Thus, we have $S = \{\mu \in X^{\oplus} \mid \mu(a) = \mu(b)\}$. Then $\varphi^{-1}(S) \cap (e + P_e^{\oplus}) = e + p^{\oplus}$ and $\varphi^{-1}(S) \cap (f + P_f^{\oplus}) = f + r^{\oplus}$. This means

$$L \cap \Psi^{-1}(S) = \pi_X(K \cap (e(X \cup p)^* \cup f(X \cup r)^*))$$

can be obtained from K using a rational transduction. More generally, whenever $\Psi^{-1}(\varphi^{-1}(S) \cap (c + P_c^{\oplus}))$ is regular for each $c \in C$, then $L \cap \Psi^{-1}(S)$ can be written as TK for a rational transduction T. This fact will be a recurring theme in the applications of Parikh annotations.

In a Parikh annotation, for each $cw \in K$ and $\mu \in P_c^{\oplus}$, we can find a word $cw' \in K$ such that $\Psi(\pi_{C\cup P}(cw')) = \Psi(\pi_{C\cup P}(cw)) + \mu$. In particular, this means $\Psi(\pi_X(cw')) = \Psi(\pi_X(cw)) + \varphi(\mu)$. In our applications, we will need a further guarantee that provides such words, but with additional information on their structure. Such a guarantee is granted by Parikh annotations with insertion marker. Suppose we have a marker symbol $\diamond \notin X$ and a word $u \in (X \cup \{\diamond\})^*$ with $u = u_0 \diamond u_1 \cdots \diamond u_n$ for $u_0, \ldots, u_n \in X^*$. Then we write $u \preceq_\diamond v$ if we have $v = u_0 v_1 u_1 \cdots v_n u_n$ for some $v_1, \ldots, v_n \in X^*$. In other words, v can be obtained from u by replacing each occurrence of $\diamond$ with some word from X^*.

Definition 9.2.3. *Let* $L \subseteq X^*$ *be a language and* $\mathcal{C}$ *be a language class. A* Parikh annotation with insertion marker (PAIM) for L in $\mathcal{C}$ *is a tuple* $(K, C, P, (P_c)_{c\in C}, \varphi, \diamond)$ *such that:*

1. $\diamond \notin X$ *and* $K \subseteq C(X \cup P \cup \{\diamond\})^*$ *is in* $\mathcal{C}$,
2. $(\pi_{C\cup X\cup P}(K), C, P, (P_c)_{c\in C}, \varphi)$ *is a Parikh annotation for* L *in* $\mathcal{C}$,

3. *there is a* $k \in \mathbb{N}$ *such that every* $w \in K$ *satisfies* $|w|_\diamond \leqslant k$ (boundedness)*, and*

4. *for each* $cw \in K$ *and* $\mu \in P_c^\oplus$*, there is a* $w' \in L$ *with* $\pi_{X\cup\diamond}(cw) \preceq_\diamond w'$ *and* $\Psi(w') = \Psi(\pi_X(cw)) + \varphi(\mu)$*. This property is called the* insertion property*.*

If $|C| = 1$*, then the PAIM is called* linear *and we also write* $(K, c, P_c, \varphi, \diamond)$ *for the PAIM, where* $C = \{c\}$*.*

In other words, in a PAIM, each $v \in L$ has an annotation $cw \in K$ in which a bounded number of positions is marked such that for each $\mu \in P_c^\oplus$, we can find a $v' \in L$ with $\Psi(v') = \Psi(v) + \varphi(\mu)$ such that v' is obtained from v by inserting words in corresponding positions in v. In particular, this guarantees $v \preceq v'$.

Example 9.2.4. *Let* L *and* $(K, C, P, (P_c)_{c\in C}, \varphi)$ *be as in Example 9.2.2. Furthermore, let*

$$K' = e \diamond (pab)^* c \diamond (qa)^* \cup f \diamond (rab)^* d \diamond (sb)^*.$$

Then $(K', C, P, (P_c)_{c\in C}, \varphi, \diamond)$ *is a PAIM for* L *in* Reg*. Indeed, every word in* K' *has at most two occurrences of* $\diamond$*. Moreover, if* $ew = e \diamond (pab)^m c \diamond (qa)^n \in K'$ *and* $\mu \in P_e^\oplus$*,* $\mu = k \cdot p + \ell \cdot q$*, then* $w' = (ab)^{k+m} c a^{\ell+n} \in L$ *satisfies*

$$\pi_{X\cup\diamond}(ew) = \diamond(ab)^m c \diamond a^n \preceq_\diamond (ab)^k (ab)^m c a^\ell a^n = w'$$

and clearly $\Psi(\pi_X(w')) = \Psi(\pi_X(ew)) + \varphi(\mu)$ *(and similarly for words* $fw \in K'$*).*

The main result of this section is that for each of the classes F_i and G_i, one can construct PAIM. More precisely, there is an algorithm that, given a language $L \in F_i$ or $L \in G_i$, constructs a PAIM for L in F_i or G_i, respectively.

Theorem 9.2.5. *Given* $i \in \mathbb{N}$ *and a language* L *in* F_i *(in* G_i*), one can construct a Parikh annotation with insertion marker for* L *in* F_i *(in* G_i*).*

To demonstrate how Parikh annotations can be applied, let us now prove Lemma 9.1.3, the missing piece in our method for computing downward closures. The basic idea is to first construct a PAIM for L. From this PAIM, using Corollary 2.8.3, one can construct a language $L' \supseteq L \cap \Psi^{-1}(S)$ in which every word admits insertions that yield a word in $L \cap \Psi^{-1}(S)$. Here, the additional information encoded into each word by the PAIM allows us to obtain L' using a rational transduction from the PAIM, which guarantees that L' is also in G_i.

Proof of Lemma 9.1.3. First, we construct a PAIM $(K, C, P, (P_c)_{c\in C}, \varphi, \diamond)$ for L in G_i using Theorem 9.2.5. Observe that for each constant symbol $c \in C$, the sets $S_c = \{\mu \in P_c^\oplus \mid \varphi(c + \mu) \in S\}$ are Presburger definable and hence effectively semilinear. Moreover, by Corollary 2.8.3, we can effectively construct a finite automaton for $\Psi^{-1}(S_c{\downarrow})$, meaning that the language

$$L' = \{\pi_X(cv) \mid c \in C,\ cv \in K,\ \pi_{P_c}(v) \in \Psi^{-1}(S_c{\downarrow})\}.$$

can be obtained from K using a rational transduction and thus effectively belongs to G_i, since G_i is an effective full semi-AFL. Let us prove that this language satisfies our requirement. The counting property of the PAIM entails that $L \cap \Psi^{-1}(S) \subseteq L'$. In order to show $L' \subseteq (L \cap \Psi^{-1}(S)){\downarrow}$, suppose we have $w \in L'$. Then there is a $cv \in K$ with $w = \pi_X(cv)$ and $\pi_{P_c}(v) \in \Psi^{-1}(S_c{\downarrow})$. This

means there is a $v \in P_c^{\oplus}$ with $\Psi(\pi_{P_c}(v)) + v \in S_c$. The insertion property of $(K, C, P, (P_c)_{c\in C}, \varphi, \diamond)$ allows us to find a word $v' \in L$ such that

$$\Psi(v') = \Psi(\pi_X(cv)) + \varphi(v), \qquad \pi_{X\cup\{\diamond\}}(cv) \preceq_\diamond v'. \tag{9.1}$$

By definition of S_c, the first part of Eq. (9.1) implies that $\Psi(v') \in S$. The second part of Eq. (9.1) means in particular that $w = \pi_X(cv) \preceq v'$. Therefore, we have $w \preceq v' \in L \cap \Psi^{-1}(S)$. □

Outline of the proof The rest of this section is devoted to the proof of Theorem 9.2.5. The construction of PAIM proceeds recursively with respect to the level of our hierarchy. This means, we show that if PAIM can be constructed for F_i, then we can compute them for G_i (Lemma 9.2.19) and if they can be constructed for G_i, then they can be computed for F_{i+1} (Lemma 9.2.20). While the latter can be done with a direct construction, the former requires a series of steps. The general idea is to use recursion with respect to the number of nonterminals via the van Leeuwen decomposition (see Section 2.6). This leaves us with two tasks:

- We construct PAIM for languages generated by one-nonterminal grammars where we are given PAIM for the right-hand-sides (Lemma 9.2.18).
- We construct PAIM for languages $\sigma(L)$, where σ is a substitution, a PAIM is given for L and for each $\sigma(x)$ (Lemma 9.2.16). This construction is again divided into the case where σ is a letter substitution (i.e., one in which each symbol is mapped to a set of letters) (Lemma 9.2.15) and the general case (Lemma 9.2.16).

Maybe surprisingly, the most conceptually involved step in the construction of PAIM lies within obtaining a Parikh annotation for $\sigma(L)$ in $\mathsf{Alg}(\mathcal{C})$, where σ is a letter substitution and a PAIM for $L \subseteq X^*$ in $\mathsf{Alg}(\mathcal{C})$ is given. Before turning to this case in Section 9.2.2, let us get a few simple cases out of the way.

9.2.1 Simple constructions

Here, we present some simple cases of the construction of Parikh annotations with insertion markers. The proofs of Lemmas 9.2.6 to 9.2.11 are not difficult, but we include them for the sake of completeness.

Lemma 9.2.6 (Finite languages). *Given L in F_0, one can construct a PAIM for L in F_0.*

Proof. Let $L = \{w_1, \dots, w_n\} \subseteq X^*$ and define $C = \{c_1, \dots, c_n\}$ and $P = P_c = \emptyset$, where the c_i are new symbols. Let $\varphi\colon (C \cup P)^{\oplus} \to X^{\oplus}$ be the morphism with $\varphi(c_i) = \Psi(w_i)$. It is easily verified that with $K = \{c_1 w_1, \dots, c_n w_n\}$, the tuple $(K, C, P, (P_c)_{c\in C}, \varphi, \diamond)$ is a PAIM for L in F_0. □

Lemma 9.2.7 (Unions). *Given $i \in \mathbb{N}$ and languages $L_0, L_1 \in G_i$, along with a PAIM in G_i for each of them, one can construct a PAIM for $L_0 \cup L_1$ in G_i.*

Proof. One can find a PAIM $(K^{(i)}, C^{(i)}, P^{(i)}, (P_c^{(i)})_{c\in C^{(i)}}, \varphi^{(i)}, \diamond)$ for L_i in $\mathcal{C}$ for $i = 0, 1$ such that $C^{(0)} \cap C^{(1)} = P^{(0)} \cap P^{(1)} = \emptyset$. Then $K = K^{(0)} \cup K^{(1)}$ is effectively contained in G_i and can be turned into a PAIM $(K, C, P, (P_c)_{c\in C}, \varphi, \diamond)$ for $L_0 \cup L_1$. □

Lemma 9.2.8 (Homomorphic images)**.** *Let* $h\colon X^* \to Y^*$ *be a morphism. Given* $i \in \mathbb{N}$ *and a PAIM for* $L \in \mathsf{G}_i$ *in* G_i*, one can construct a PAIM for* $h(L)$ *in* G_i*.*

Proof. Let $(K, C, P, (P_c)_{c\in C}, \varphi, \diamond)$ be a PAIM for L and let $\bar{h}\colon X^\oplus \to Y^\oplus$ be the morphism with $\bar{h}(x) = \Psi(h(x))$ for $x \in X$. We choose $\varphi'\colon (C \cup P)^\oplus \to Y^\oplus$ to be the morphism with $\varphi'(\mu) = \bar{h}(\varphi(\mu))$ for $\mu \in (C \cup P)^\oplus$. Moreover, let

$$g\colon (C \cup X \cup P \cup \{\diamond\})^* \to (C \cup Y \cup P \cup \{\diamond\})^*$$

be the extension of h that fixes $C \cup P \cup \{\diamond\}$. Then $(g(K), C, P, (P_c)_{c\in C}, \varphi', \diamond)$ is clearly a PAIM for $h(L)$ in G_i. □

Lemma 9.2.9 (Linear decomposition)**.** *Given* $i \in \mathbb{N}$ *and* $L \in \mathsf{G}_i$ *along with a PAIM in* G_i*, one can construct* $L_1, \ldots, L_n \in \mathsf{G}_i$*, each together with a linear PAIM in* G_i*, such that* $L = L_1 \cup \cdots \cup L_n$*.*

Proof. Let $(K, C, P, (P_c)_{c\in C}, \varphi, \diamond)$ be a PAIM for $L \subseteq X^*$. For each $c \in C$, let

$$K_c = K \cap c(X \cup P \cup \{\diamond\})^*.$$

Then $(K_c, \{c\}, P_c, P_c, \varphi_c, \diamond)$, where φ_c is the restriction of φ to $(\{c\} \cup P_c)^\oplus$, is a PAIM for $\pi_X(K_c)$ in G_i. Furthermore, $L = \bigcup_{c\in C} \pi_X(K_c)$. □

Lemma 9.2.10 (Presence check)**.** *Let* X *be an alphabet and* $x \in X$*. Given* $i \in \mathbb{N}$ *and a PAIM for* $L \subseteq X^*$ *in* G_i*, one can construct a PAIM for* $L \cap X^*xX^*$ *in* G_i*.*

Proof. Since

$$(L_1 \cup \cdots \cup L_n) \cap X^*xX^* = (L_1 \cap X^*xX^*) \cup \cdots \cup (L_n \cap X^*xX^*),$$

Lemma 9.2.9 and Lemma 9.2.7 imply that we may assume that for L, we have a linear PAIM $(K, c, P_c, \varphi, \diamond)$. Since in the case $\varphi(c)(x) \geqslant 1$, we have the equation $L \cap X^*xX^* = L$ and there is nothing to do, we assume $\varphi(c)(x) = 0$.

Let $C' = \{(c, p) \mid p \in P, \varphi(p)(x) \geqslant 1\}$ be a new alphabet and let

$$K' = \{(c, p)uv \mid (c, p) \in C',\ u, v \in (X \cup P \cup \{\diamond\})^*,\ cupv \in K\}.$$

Note that K' can clearly be obtained from K by way of a rational transduction and is therefore contained in G_i. Furthermore, we let $P' = P'_{(c,p)} = P$ and $\varphi'((c, p)) = \varphi(c) + \varphi(p)$ for $(c, p) \in C'$ and $\varphi'(p) = \varphi(p)$ for $p \in P$. Then we have

$$\begin{aligned}\pi_X(K') &= \{\pi_X(w) \mid w \in K,\ \exists p \in P : \varphi(\pi_{C\cup P}(w))(p) \geqslant 1, \varphi(p)(x) \geqslant 1\}\\ &= \{\pi_X(w) \mid w \in K,\ |\pi_X(w)|_x \geqslant 1\} = L \cap X^*xX^*.\end{aligned}$$

This proves the projection property. For each $(c, p)uv \in K'$ with $cupv \in K$, we have

$$\varphi'(\pi_{C'\cup P'}((c, p)uv)) = \varphi(\pi_{C\cup P}(cupv)) = \Psi(\pi_X(cupv)) = \Psi(\pi_X((c, p)uv)).$$

and thus $\varphi'(\pi_{C'\cup P'}(w)) = \Psi(\pi_X(w))$ for every $w \in K'$. Hence, we have established the counting property. Moreover,

$$\Psi(\pi_{C'\cup P'}(K')) = \bigcup_{p\in P} (c, p) + P'^\oplus,$$

meaning the commutative projection property is satisfied as well. This proves that the tuple $(\pi_{C\cup X\cup P}(K'), C', P', (P'_d)_{d\in C'}, \varphi')$ is a Parikh annotation for the language $L \cap X^*xX^*$ in G_i. Since $(K, C, P, (P_c)_{c\in C}, \varphi, \diamond)$ is a PAIM for L, it follows that the tuple $(K', C', P', (P'_d)_{d\in C'}, \varphi', \diamond)$ is a PAIM for $L \cap X^*xX^*$. □

Lemma 9.2.11 (Absence check)**.** *Let X be an alphabet and $x \in X$. Given $i \in \mathbb{N}$ and a PAIM for $L \subseteq X^*$ in G_i, one can construct a PAIM for $L \setminus X^*xX^*$ in G_i.*

Proof. Since

$$(L_1 \cup \cdots \cup L_n) \setminus X^*xX^* = (L_1 \setminus X^*xX^*) \cup \cdots \cup (L_n \setminus X^*xX^*),$$

Lemma 9.2.9 and Lemma 9.2.7 imply that we may assume that for L, we are given a linear PAIM $(K, c, P_c, \varphi, \diamond)$. Since in the case $\varphi(c)(x) \geqslant 1$, we have $L \setminus X^*xX^* = \emptyset$ and there is nothing to do, we assume $\varphi(c)(x) = 0$.

Let $C' = C$, $P' = P'_c = \{p \in P \mid \varphi(p)(x) = 0\}$, and let

$$K' = \{w \in K \mid |w|_p = 0 \text{ for each } p \in P \setminus P'\}.$$

Furthermore, we let φ' be the restriction of φ to $(C' \cup P')^{\oplus}$. Then it is clear that $(K', C', (P'_c)_{c\in C'}, \varphi', \diamond)$ is a PAIM for $L \setminus X^*xX^*$ in G_i. □

9.2.2 Substitutions

In this section, we construct PAIM for languages $\sigma(L)$, where $L \in G_i$, σ is a G_i-substitution. As mentioned above, the main obstacle in the construction of PAIM consists of obtaining one for $\sigma(L)$, where σ is a letter substitution and we are given a PAIM for L. Of course, the basic idea is to replace symbols in the PAIM for L in the same way as σ does. However, one has to substitute the symbols in X consistently with the symbols in $C \cup P$; more precisely, one has to maintain the agreement between $\varphi(\pi_{C\cup P}(\cdot))$ and $\Psi(\pi_X(\cdot))$.

In order to exploit the fact that this agreement exists in the first place, we use the following simple yet very useful lemma. It states that for a morphism ψ into a group, the only way a grammar G can guarantee $L(G) \subseteq \psi^{-1}(h)$ is by encoding into each nonterminal A the value $\psi(u)$ for the words u that A derives. The G-compatible extension of ψ reconstructs this value for each nonterminal. Let $G = (N, T, P, S)$ be a $\mathcal{C}$-grammar and M be a monoid. A morphism $\psi\colon (N \cup T)^* \to M$ is called G-*compatible* if $u \Rightarrow^*_G v$ implies that $\psi(u) = \psi(v)$ for $u, v \in (N \cup T)^*$. We will essentially apply the following lemma by regarding $X^{\oplus}$ as a subset of $\mathbb{Z}^n$ and defining $\psi\colon (C \cup P \cup X)^* \to \mathbb{Z}^n$ as the morphism with $\psi(w) = \Psi(\pi_X(w)) - \varphi(\pi_{C\cup P}(w))$. In the case that G generates the corresponding Parikh annotation, the counting property implies that $L(G) \subseteq \psi^{-1}(0)$. The lemma then states that each nonterminal in G encodes the imbalance between $\Psi(\pi_X(\cdot))$ and $\varphi(\pi_{C\cup P}(\cdot))$ on the words it generates.

Lemma 9.2.12. *Let H be a group and $\psi\colon T^* \to H$ be a morphism. Furthermore, let $G = (N, T, P, S)$ be a reduced $\mathcal{C}$-grammar with $L(G) \subseteq \psi^{-1}(h)$ for some $h \in H$. Then ψ has a unique G-compatible extension $\hat{\psi}\colon (N \cup T)^* \to H$.*

Proof. First, observe that there is at most one G-compatible extension: For each $A \in N$, there is a $u \in T^*$ with $A \Rightarrow^*_G u$ and hence $\hat{\psi}(A) = \psi(u)$.

In order to prove existence, we claim that for each $A \in N$ and $A \Rightarrow_G^* u$ and $A \Rightarrow_G^* v$ for $u, v \in T^*$, we have $\psi(u) = \psi(v)$. Indeed, since G is reduced, there are $x, y \in T^*$ with $S \Rightarrow_G^* xAy$. Then xuy and xvy are both in $L(G)$ and hence $\psi(xuy) = \psi(xvy) = h$. In the group H, this implies

$$\psi(u) = \psi(x)^{-1} h \psi(y)^{-1} = \psi(v).$$

This means a G-compatible extension exists: Setting $\hat{\psi}(A) = \psi(w)$ for some $w \in T^*$ with $A \Rightarrow_G^* w$ does not depend on the chosen w. This definition implies that whenever $u \Rightarrow_G^* v$ for $u \in (N \cup T)^*$, $v \in T^*$, we have $\hat{\psi}(u) = \hat{\psi}(v)$. Therefore, if $u \Rightarrow_G^* v$ for $u, v \in (N \cup T)^*$, picking a $w \in T^*$ with $v \Rightarrow_G^* w$ yields $\hat{\psi}(u) = \hat{\psi}(w) = \hat{\psi}(v)$. Hence, $\hat{\psi}$ is G-compatible. □

Note that in the case $\mathcal{C} = F_i$ and $H = \mathbb{Z}^n$, the G-compatible extension can be computed: Since H is commutative, ψ is well-defined on $(N \cup T)^{\oplus}$ (i.e., there is a morphism $\bar{\psi} \colon (N \cup T)^{\oplus} \to \mathbb{Z}$ with $\bar{\psi}(\Psi(w)) = \psi(w)$ for $w \in (N \cup T)^*$) and we can determine $\hat{\psi}(A)$ by computing a semilinear representation of the language $\{w \in (N \cup T)^* \mid A \Rightarrow_G^* w\}$, which is in $\mathsf{Alg}(F_i)$.

We continue with the problem of replacing $C \cup P$ and X consistently. For constructing the PAIM for $\sigma(L)$, it is easy to see that it suffices to consider the case where $\sigma(a) = \{a, b\}$ for some $a \in X$ and $\sigma(x) = \{x\}$ for $x \in X \setminus \{a\}$.

In order to simplify the setting and exploit the symmetry of the roles played by $C \cup P$ and X, we consider a slightly more general situation. There is an alphabet $X = X_0 \uplus X_1$, morphisms $\gamma_i \colon X_i^* \to \mathbb{N}$, $i = 0, 1$, and a language $L \subseteq X^*$, $L \in \mathsf{Alg}(F_i)$ with $\gamma_0(\pi_{X_0}(w)) = \gamma_1(\pi_{X_1}(w))$ for every $w \in L$. Roughly speaking, X_1 will later play the role of $C \cup P$ and X_0 will play the role of X. Then, $\gamma_0(w)$ will be the number of a's in w and $\gamma_1(w)$ will be the number of a's represented by symbols from $C \cup P$ in w. Therefore, we wish to construct a language L' in $\mathsf{Alg}(F_i)$ such that each word in L' is obtained from a word in L as follows. We substitute each occurrence of $x \in X_i$ by one of $\gamma_i(x) + 1$ many symbols y in an alphabet Y_i, each of which will be assigned a value $0 \leqslant \eta_i(y) \leqslant \gamma_i(x)$. Here, we want to guarantee that in every resulting word $w \in (Y_0 \cup Y_1)^*$, we have $\eta_0(\pi_{Y_0}(w)) = \eta_1(\pi_{Y_1}(w))$, meaning that the symbols in X_0 and in X_1 are replaced *consistently*. Formally, we have

$$Y_i = \{(x, j) \mid x \in X_i,\ 0 \leqslant j \leqslant \gamma_i(x)\},\ \ i = 0, 1, \quad Y = Y_0 \cup Y_1 \tag{9.2}$$

and the morphisms

$$\begin{array}{lll} h_i \colon Y_i^* \longrightarrow X_i^*, & h \colon Y^* \longrightarrow X^*, & \eta_i \colon Y_i^* \longrightarrow \mathbb{N}, \\ (x, j) \longmapsto x, & (x, j) \longmapsto x, & (x, j) \longmapsto j, \end{array} \tag{9.3}$$

and we want to construct a subset of

$$\hat{L} = \{w \in h^{-1}(L) \mid \eta_0(\pi_{Y_0}(w)) = \eta_1(\pi_{Y_1}(w))\}$$

in $\mathsf{Alg}(\mathcal{C})$. Observe that we cannot hope to find $\hat{L}$ itself in $\mathsf{Alg}(\mathcal{C})$ in general. Take, for example, the context-free language $E = \{a^n b^n \mid n \geqslant 0\}$ and $X_0 = \{a\}$, $X_1 = \{b\}$, $\gamma_0(a) = 1$, $\gamma_1(b) = 1$. Then we would have

$$\hat{E} \cap (a, 0)^*(a, 1)^*(b, 0)^*(b, 1)^* = \{(a, 0)^m (a, 1)^n (b, 0)^m (b, 1)^n \mid m, n \geqslant 0\},$$

which is clearly not context-free. However, the language

$$E' = \{wg(w)^R \mid w \in \{(a,0),(a,1)\}^*\},$$

where $g\colon \{(a,0),(a,1)\}^* \to \{(b,0),(b,1)\}^*$ is the morphism with $g((a,j)) = (b,j)$ for $j = 0,1$, is context-free. Although it is only a proper subset of $\hat{E}$, it is large enough to satisfy $\pi_{Y_i}(E') = \pi_{Y_i}(\hat{E}) = \pi_{Y_i}(h^{-1}(E))$ for $i = 0,1$. We will see that in order to construct Parikh annotations, it suffices to use such under-approximations of $\hat{L}$. Proposition 9.2.14 will provide us with a general method to compute them. First, we need a combinatorial lemma (Lemma 9.2.13), whose formulation requires the concepts of derivation trees and matchings.

Derivation trees and matchings By an *X-labeled tree*, we mean a finite ordered unranked tree in which each node carries a label from $X \cup \{\varepsilon\}$ for an alphabet X. For each node, there is a linear order on the set of its children. For each node x, we write $c(x) \in X^*$ for the word obtained by reading the labels of x's children in this order. Furthermore, $\mathsf{yield}(x) \in X^*$ denotes the word obtained by reading leaf labels below the node x according to the linear order induced on the leaves. Moreover, if r is the root of t, we also write $\mathsf{yield}(t)$ for $\mathsf{yield}(r)$. The *height* of a tree is the maximal length of a path from the root to a leaf, i.e. a tree consisting of a single node has height 0. A *subtree* of a tree t is the tree consisting of all nodes below some node x of t. If x is a child of t's root, the subtree is a *direct subtree*.

Let $G = (N,T,P,S)$ be a $\mathcal{C}$-grammar. A *partial derivation tree (for G)* is an $(N \cup T)$-labeled tree t in which (i) each inner node x has a label $A \in N$ and there is some $A \to L$ in P with $c(x) \in L$, and (ii) no ε-labeled node has a sibling. If, in addition, the root is labeled S and every leaf is labeled by $T \cup \{\varepsilon\}$, it is called a *derivation tree for* G.

Let t be a tree whose leaves are $X \cup \{\varepsilon\}$-labeled. Let L_i denote the set of X_i-labeled leaves of t. An *arrow collection for* t is a finite set A together with maps $\nu_i\colon A \to L_i$ for $i = 0,1$. Hence, A can be thought of as a set of arrows pointing from X_0-labeled leaves to X_1-labeled leaves. We say an arrow $a \in A$ is *incident* to a leaf ℓ if $\nu_0(a) = \ell$ or $\nu_1(a) = \ell$. If ℓ is a leaf, then $d_A(\ell)$ denotes the number of arrows incident to ℓ. More generally, for a subtree s of t, $d_A(s)$ denotes the number of arrows incident to some leaf in s and some leaf outside of s. A is called a k-*matching* if

1. each leaf labeled $x \in X_i$ has precisely $\gamma_i(x)$ incident arrows, and
2. $d_A(s) \leqslant k$ for every subtree s of t.

Here, the name *matching* stems from the fact that one can think of each symbol $x \in X_i$ as representing $\gamma_i(x)$ items of some kind. Of course, the condition $\gamma_0(w) = \gamma_1(w)$ then states that in w, the same number of items is represented by symbols in X_0 as by symbols in X_1. In this vein, a k-matching can be thought of as a one-to-one correspondence between the items represented by symbols in X_0 and those represented by symbols in X_1. The second condition above then expresses that in each subtree s, at most k items are in correspondence with items outside of s.

The following lemma applies Lemma 9.2.12. The latter tells us that for nodes x of a derivation tree, the balance $\gamma_0(\pi_{X_0}(\mathsf{yield}(x))) - \gamma_1(\pi_{X_1}(\mathsf{yield}(x)))$ is bounded. This can be used to construct k-matchings in a bottom-up manner.

Lemma 9.2.13. *Let* $X = X_0 \uplus X_1$ *and* $\gamma_i \colon X_i^* \to \mathbb{N}$ *for* $i = 0, 1$ *be a morphism. Let* G *be a reduced* F_i*-grammar with* $L(G) \subseteq X^*$ *and* $\gamma_0(\pi_{X_0}(w)) = \gamma_1(\pi_{X_1}(w))$ *for every* $w \in L(G)$*. Then one can compute a bound* $k \in \mathbb{N}$ *such that each derivation tree of* G *admits a* k*-matching.*

Proof. Let $G = (N, X, P, S)$ and let $\delta \colon X^* \to \mathbb{Z}$ be the morphism with

$$\delta(w) = \gamma_0(\pi_{X_0}(w)) - \gamma_1(\pi_{X_1}(w))$$

for $w \in X^*$. Since then $\delta(w) = 0$ for every $w \in L(G)$, by Lemma 9.2.12, δ extends uniquely to a G-compatible $\hat{\delta} \colon (N \cup X)^* \to \mathbb{Z}$. We claim that for the choice $k = \max\{|\hat{\delta}(A)| \mid A \in N\}$, each derivation tree of G admits a k-matching.

Consider an $(N \cup X)$-tree t and let L_i be the set of X_i-labeled leaves. Let A be an arrow collection for t and let $d_A(\ell)$ be the number of arrows incident to $\ell \in L_0 \cup L_1$. Moreover, let $\lambda(\ell)$ be the label of the leaf ℓ and let

$$\beta(t) = \sum_{\ell \in L_0} \gamma_0(\lambda(\ell)) - \sum_{\ell \in L_1} \gamma_1(\lambda(\ell)).$$

A is a *partial* k*-matching* if the following holds:

1. if $\beta(t) \geqslant 0$, then $d_A(\ell) \leqslant \gamma_0(\lambda(\ell))$ for each $\ell \in L_0$ and $d_A(\ell) = \gamma_1(\lambda(\ell)))$ for each $\ell \in L_1$.
2. if $\beta(t) \leqslant 0$, then $d_A(\ell) \leqslant \gamma_1(\lambda(\ell))$ for each $\ell \in L_1$ and $d_A(\ell) = \gamma_0(\lambda(\ell)))$ for each $\ell \in L_0$.
3. $d_A(s) \leqslant k$ for every subtree s of t.

Hence, while in a k-matching the number $\gamma_i(\lambda(\ell))$ is the degree of ℓ (with respect to the matching), it is merely a capacity in a partial k-matching. The first two conditions express that either all leaves in L_0 or all in L_1 (or both) are filled up to capacity, depending on which of the two sets of leaves has less (total) capacity.

If t is a derivation tree of G, then $\beta(t) = 0$ and hence a partial k-matching is already a k-matching. Therefore, we show by induction on n that every derivation subtree of height n admits a partial k-matching. This is trivial for $n = 0$ and for $n > 0$, consider a derivation subtree t with direct subtrees $s_1, \ldots, s_r$. Let B be the label of t's root and $B_j \in N \cup X$ be the label of s_j's root. Then $\hat{\delta}(B) = \beta(t)$, $\hat{\delta}(B_j) = \beta(s_j)$ and $\beta(t) = \sum_{j=1}^{r} \beta(s_j)$. By induction, each s_j admits a partial k-matching A_j. Let A be the union of the A_j. Observe that since $\sum_{\ell \in L_0} d_A(\ell) = \sum_{\ell \in L_1} d_A(\ell)$ in every arrow collection (each side equals the number of arrows), we have

$$\beta(t) = \underbrace{\sum_{\ell \in L_0} (\gamma_0(\lambda(\ell)) - d_A(\ell))}_{=:p \geqslant 0} - \underbrace{\sum_{\ell \in L_1} (\gamma_1(\lambda(\ell)) - d_A(\ell))}_{=:q \geqslant 0}. \tag{9.4}$$

If $\beta(t) \geqslant 0$ and hence $p \geqslant q$, this equation allows us to obtain A' from A by adding q arrows, such that each $\ell \in L_1$ has $\gamma_1(\lambda(\ell)) - d_A(\ell)$ new incident arrows. They are connected to X_0-leaves so as to maintain $\gamma_0(\ell) - d_{A'}(\ell) \geqslant 0$. Symmetrically, if $\beta(t) \leqslant 0$ and hence $p \leqslant q$, we add p arrows such that each

$\ell \in L_0$ has $\gamma_0(\lambda(\ell)) - d_A(\ell)$ new incident arrows. They also are connected to X_1-leaves so as to maintain $\gamma_1(\lambda(\ell)) - d_{A'}(\ell) \geqslant 0$. Then by construction, A' satisfies the first two conditions of a partial k-matching. Hence, it remains to be shown that the third is fulfilled as well.

Since for each j, we have either $d_A(\ell) = \gamma_0(\lambda(\ell))$ for all $\ell \in L_0 \cap s_j$ or we have $d_A(\ell) = \gamma_1(\lambda(\ell))$ for all $\ell \in L_1 \cap s_j$, none of the new arrows can connect two leaves inside of s_j. This means the s_j are the only subtrees for which we have to verify the third condition, which amounts to checking that $d_{A'}(s_j) \leqslant k$ for $1 \leqslant j \leqslant r$. As in Eq. (9.4), we have

$$\beta(s_j) = \underbrace{\sum_{\ell \in L_0 \cap s_j} (\gamma_0(\lambda(\ell)) - d_A(\ell))}_{=:u \geqslant 0} - \underbrace{\sum_{\ell \in L_1 \cap s_j} (\gamma_1(\lambda(\ell)) - d_A(\ell))}_{=:v \geqslant 0}.$$

Since the arrows added in A' have respected the capacity of each leaf, we have $d_{A'}(s_j) \leqslant u$ and $d_{A'}(s_j) \leqslant v$. Moreover, since A_j is a partial k-matching, we have $u = 0$ or $v = 0$. In any case, we have

$$d_{A'}(s_j) \leqslant |u - v| = |\beta(s_j)| = |\widehat{\delta}(B_j)| \leqslant k,$$

proving the third condition. □

We are now ready to construct the approximations that are necessary to obtain PAIM for substitutions.

Proposition 9.2.14 (Consistent substitution). *Let* $X = X_0 \uplus X_1$ *and* $\gamma_i \colon X_i^{\oplus} \to \mathbb{N}$ *for* $i = 0, 1$ *be a morphism. Let* $L \in \mathsf{Alg}(F_i)$, $L \subseteq X^*$, *be a language with*

$$\gamma_0(\pi_{X_0}(w)) = \gamma_1(\pi_{X_1}(w))$$

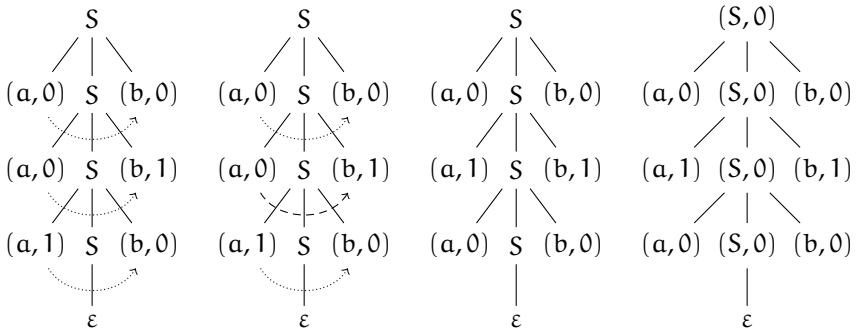
for every $w \in L$. *Furthermore, let* Y_i, h_i, η_i *for* $i = 0, 1$ *and* Y, h *be defined as in Eq. (9.2) and Eq. (9.3). Moreover, let* L *be given by a reduced grammar. Then one can construct a language* $L' \in \mathsf{Alg}(F_i)$, $L' \subseteq Y^*$, *with*

1. $L' \subseteq h^{-1}(L)$,
2. $\pi_{Y_i}(L') = \pi_{Y_i}(h^{-1}(L))$ *for* $i = 0, 1$,
3. $\eta_0(\pi_{Y_0}(w)) = \eta_1(\pi_{Y_1}(w))$ *for every* $w \in L'$.

Proof. Let $G_0 = (N, X, P_0, S)$ be a reduced F_i-grammar with $L(G_0) = L$. Let $G_1 = (N, Y, P_1, S)$ be the grammar with $P_1 = \{A \to \widehat{h}^{-1}(K) \mid A \to K \in P_0\}$, where $\widehat{h} \colon (N \cup Y)^* \to (N \cup X)^*$ is the extension of $h \colon Y^* \to X^*$ that fixes N. With $L_1 = L(G_1)$, we clearly have $L_1 = h^{-1}(L)$.

According to Lemma 9.2.13, we can find a $k \in \mathbb{N}$ such that every derivation tree of G_0 admits a k-matching. With this, let $F = \{z \in \mathbb{Z} \mid |z| \leqslant k\}$, $N_2 = N \times F$, and η be the morphism

$$\begin{aligned} \eta \colon (N_2 \cup Y)^* &\longrightarrow \mathbb{Z}, \\ (A, z) &\longmapsto z && \text{for } (A, z) \in N_2, \\ y &\longmapsto \eta_0(\pi_{Y_0}(y)) - \eta_1(\pi_{Y_1}(y)) && \text{for } y \in Y. \end{aligned}$$



(a) t; arrows in A (b) t; $i = 1$; dashed arrow is the one in A' (c) t' (d) t''

Figure 9.1: Derivation trees in the proof of Proposition 9.2.14 for the context-free grammar G with productions $S \to aSb$, $S \to \varepsilon$ and $X_0 = \{a\}$, $X_1 = \{b\}$, $\gamma_0(a) = \gamma_1(b) = 1$.

Moreover, let $g\colon (N_2 \cup Y)^* \to (N \cup Y)^*$ be the morphism with $g((A,z)) = A$ and $g(y) = y$ for $y \in Y$. This allows us to define the set of productions

$$P_2 = \{(A,z) \to g^{-1}(K) \cap \eta^{-1}(z) \mid A \to K \in P_1\}.$$

Note that since F_i is an effective Presburger closed full semi-trio, we have effectively $g^{-1}(K) \cap \eta^{-1}(z) \in \mathsf{F}_i$ for $K \in \mathsf{F}_i$. Finally, let G_2 be the grammar $G_2 = (N_2, Y, P_2, (S,0))$. We claim that $L' = L(G_2)$ has the desired properties. Since $L' \subseteq L_1 = h^{-1}(L)$, condition 1 is satisfied. Furthermore, the construction guarantees that for a production $(A,z) \to w$ in G_2, we have $\eta(w) = z$. In particular, every $w \in Y^*$ with $(S,0) \Rightarrow_{G_2}^* w$ exhibits

$$\eta_0(\pi_{Y_0}(w)) - \eta_1(\pi_{Y_1}(w)) = \eta(w) = 0.$$

Thus, we have shown condition 3.

Note that the inclusion "$\subseteq$" of condition 2 follows from condition 1. In order to prove the inclusion "$\supseteq$", we shall use k-matchings in G_0 to construct derivations in G_2. See Fig. 9.1 for an example of the following construction of derivation trees. Let $w \in h^{-1}(L) = L(G_1)$ and consider a derivation tree t for w in G_1. Let $\bar{t}$ be the $(N \cup X)$-tree obtained from t by replacing each leaf label $y \in Y$ by $h(y)$. Then $\bar{t}$ is a derivation tree of G_0 and admits a k-matching $\bar{A}$. Since $\bar{t}$ and t are isomorphic up to labels, we can obtain a corresponding arrow collection A in t (see Fig. 9.1a).

Let L_i denote the set of Y_i-labeled leaves of t for $i = 0, 1$. Now fix $i \in \{0,1\}$. We choose a subset $A' \subseteq A$ as follows. Since $\bar{A}$ is a k-matching, each leaf $\ell \in L_i$ of t has precisely $\gamma_i(h(\lambda(\ell))) \geqslant \eta_i(\lambda(\ell))$ incident arrows in A. For each such $\ell \in L_i$, we include some arbitrary choice of $\eta_i(\lambda(\ell))$ arrows in A' (see Fig. 9.1b). The tree t' is obtained from t by changing the label of each leaf $\ell \in L_{1-i}$ from (x,j) to (x,j'), where j' is the number of arrows in A' incident to ℓ (see Fig. 9.1c). Note that since we only change labels of leaves in L_{1-i}, we have

$$\pi_{Y_i}(\mathsf{yield}(t')) = \pi_{Y_i}(\mathsf{yield}(t)) = \pi_{Y_i}(w).$$

For every subtree s of t', we define

$$\beta(s) = \eta_0(\pi_{Y_0}(\mathsf{yield}(s))) - \eta_1(\pi_{Y_1}(\mathsf{yield}(s))).$$

By construction of A', each leaf $\ell \in L_j$ has precisely $\eta_j(\lambda(\ell))$ incident arrows in A' for $j = 0, 1$. Therefore,

$$\beta(s) = \sum_{\ell \in L_0 \cap s} d_{A'}(\ell) - \sum_{\ell \in L_1 \cap s} d_{A'}(\ell). \tag{9.5}$$

The absolute value of the right-hand side of this equation is at most $d_{A'}(s)$ and hence

$$|\eta_0(\pi_{Y_0}(\mathsf{yield}(s))) - \eta_1(\pi_{Y_1}(\mathsf{yield}(s)))| = |\beta(s)| \leqslant d_{A'}(s) \leqslant d_A(s) \leqslant k \tag{9.6}$$

since $\bar{A}$ is a k-matching. In the case $s = t'$, Eq. (9.5) also tells us that

$$\eta_0(\pi_{Y_0}(\mathsf{yield}(t'))) - \eta_1(\pi_{Y_1}(\mathsf{yield}(t'))) = \sum_{\ell \in L_0} d_{A'}(\ell) - \sum_{\ell \in L_1} d_{A'}(\ell) = 0. \tag{9.7}$$

Let t'' be the tree obtained from t' as follows: For each N-labeled node x of t', we replace the label B of x with $(B, \beta(s))$, where s is the subtree below x (see Fig. 9.1d). By Eq. (9.6), this is a symbol in N_2. The root node of t'' has label $(S, 0)$ by Eq. (9.7). Furthermore, it follows by an induction on the height of subtrees that if (B, z) is the label of a node x, then $z = \eta(c(x))$. Hence, the tree t'' is a derivation tree of G_2. This means

$$\pi_{Y_i}(w) = \pi_{Y_i}(\mathsf{yield}(t')) = \pi_{Y_i}(\mathsf{yield}(t'')) \in L(G_2) = L',$$

completing the proof of condition 2. □

Proposition 9.2.14 now allows us to construct PAIM for languages $\sigma(L)$, where σ is a letter substitution. The essential idea of Lemma 9.2.15's proof is to use a PAIM $(K, C, P, (P_c)_{c \in C}, \varphi, \diamond)$ for L and then apply Proposition 9.2.14 to K with $X_0 = Z \cup \{\diamond\}$ and $X_1 = C \cup P$. One can clearly assume that a single letter a from Z is replaced by $\{a, b\} \subseteq Z'$. We can therefore choose $\gamma_0(w)$ to be the number of a's in w and $\gamma_1(w)$ to be the number of a's represented by symbols in $C \cup P$. Then the counting property of K entails $\gamma_0(w) = \gamma_1(w)$ for $w \in K$ and thus applicability of Proposition 9.2.14. Condition 2 then yields the projection property for $i = 0$ and the commutative projection property for $i = 1$ and condition 3 yields the counting property for the new PAIM.

Lemma 9.2.15 (Letter substitution). *Let $\sigma\colon Z \to \mathcal{P}(Z')$ be a letter substitution. Given $i \in \mathbb{N}$ and a PAIM for $L \in G_i$ in G_i, one can construct a PAIM in G_i for $\sigma(L)$.*

Proof. In light of Lemma 9.2.8, it clearly suffices to prove the statement in the case that there are $a \in Z$ and $b \in Z'$ with $Z' = Z \cup \{b\}$, $b \notin Z$ and $\sigma(x) = \{x\}$ for $x \in Z \setminus \{a\}$ and $\sigma(a) = \{a, b\}$. Let $(K, C, P, (P_c)_{c \in C}, \varphi, \diamond)$ be a PAIM for L in G_i. We may clearly assume K to be given by a reduced F_i-grammar.

We want to use Proposition 9.2.14 to construct a PAIM for $\sigma(L)$. To this end, we define $X_0 = Z \cup \{\diamond\}$ and $X_1 = C \cup P$. Furthermore, let $\gamma_i\colon X_i^* \to \mathbb{N}$ for $i = 0, 1$ be the morphisms with

$$\gamma_0(w) = |w|_a, \quad \gamma_1(w) = \varphi(w)(a).$$

Then, by the counting property of PAIM, we have $\gamma_0(w) = \gamma_1(w)$ for each $w \in K$. Let Y, h and Y_i, h_i, η_i be defined as in Eq. (9.2) and Eq. (9.3). Now Proposition 9.2.14 allows us to construct a language $\hat{K} \in G_i$, $\hat{K} \subseteq Y^*$, with $\hat{K} \subseteq h^{-1}(K)$ and

$$\begin{aligned} \pi_{X_i}(\hat{K}) &= \pi_{X_i}(h^{-1}(K)) && \text{for } i = 0, 1, \\ \eta_0(\pi_{X_0}(w)) &= \eta_1(\pi_{X_1}(w)) && \text{for } w \in \hat{K}. \end{aligned}$$

For each $f \in C \cup P$, let $D_f = \{(f', m) \in Y_1 \mid f' = f\}$. With this, define $C' = \bigcup_{c \in C} D_c$, $P' = \bigcup_{p \in P} D_p$, and $P'_{(c,m)} = \bigcup_{p \in P_c} D_p$ for $(c, m) \in C'$. The new morphism $\varphi' \colon (C' \cup P')^{\oplus} \to Z'^{\oplus}$ is defined by

$$\begin{aligned} \varphi'((f, m))(z) &= \varphi(f)(z) && \text{for } z \in Z \setminus \{a\}, \\ \varphi'((f, m))(b) &= m, \\ \varphi'((f, m))(a) &= \varphi(f)(a) - m. \end{aligned}$$

Let $g \colon Y^* \to (C' \cup Z' \cup P' \cup \{\diamond\})^*$ be the morphism with $g((z, 0)) = z$ for $z \in Z$, $g((a, 1)) = b$, $g(x) = x$ for $x \in C' \cup P' \cup \{\diamond\}$. We claim that with $K' = g(\hat{K})$, the tuple $(K', C', P', (P'_c)_{c \in C'}, \varphi', \diamond)$ is a PAIM for $\sigma(L)$. First, note that $K' \in G_i$ and

$$K' = g(\hat{K}) \subseteq g(h^{-1}(K)) \subseteq g(h^{-1}(C(Z \cup P)^*)) \subseteq C'(Z' \cup P')^*.$$

Note that g is bijective. We can therefore define

$$f \colon (C' \cup Z' \cup P' \cup \{\diamond\})^* \to (C \cup Z \cup P \cup \{\diamond\})^*$$

to be the morphism with $f(w) = h(g^{-1}(w))$ for all w. Observe that then we have $f(a) = f(b) = a$ and $f(z) = z$ for $z \in Z \setminus \{a, b\}$. Moreover, by the definition of K', we have $f(K') \subseteq K$ and $\sigma(L) = f^{-1}(L)$.

- *Projection property.* Note that whenever we have $\pi_{Y_0}(u) = \pi_{Y_0}(v)$ for words $u, v \in Y^*$, we also have $\pi_{Z'}(g(u)) = \pi_{Z'}(g(v))$. Thus, from the equality $\pi_{Y_0}(\hat{K}) = \pi_{Y_0}(h^{-1}(K))$, we deduce

$$\begin{aligned} \pi_{Z'}(K') &= \pi_{Z'}(g(\hat{K})) = \pi_{Z'}(g(h^{-1}(K))) \\ &= \pi_{Z'}(f^{-1}(K)) = f^{-1}(L) = \sigma(L). \end{aligned}$$

- *Counting property.* Note that by the definition of φ' and g, we have

$$\varphi'(\pi_{C' \cup P'}(x))(b) = \eta_1(x) = \eta_1(g^{-1}(x)) \tag{9.8}$$

  for every $x \in C' \cup P'$.

  For $w \in K'$, we have $f(w) \in K$ and hence $\varphi(\pi_{C \cup P}(f(w))) = \Psi(\pi_Z(f(w)))$. Since for $z \in Z \setminus \{a\}$, we have $\varphi'(x)(z) = \varphi(f(x))(z)$ for every $x \in C' \cup P'$, it follows that

$$\begin{aligned} \varphi'(\pi_{C' \cup P'}(w))(z) &= \varphi(\pi_{C \cup P}(f(w)))(z) \\ &= \Psi(\pi_Z(f(w)))(z) = \Psi(\pi_{Z'}(w))(z). \end{aligned} \tag{9.9}$$

Moreover, by (9.8) and since $g^{-1}(w) \in \hat{K}$, we have

$$\begin{aligned}\varphi'(\pi_{C'\cup P'}(w))(b) &= \eta_1(g^{-1}(w)) = \eta_0(g^{-1}(w)) = |w|_b \\ &= \Psi(\pi_{Z'}(w))(b). \end{aligned} \tag{9.10}$$

and $f(w) \in K$ yields

$$\begin{aligned}\varphi'(\pi_{C'\cup P'}(w))(a) + \varphi'(\pi_{C'\cup P'}(w))(b) &= \varphi(\pi_{C\cup P}(f(w)))(a) \\ &= \Psi(\pi_Z(f(w)))(a) \\ &= \Psi(\pi_{Z'}(w))(a) + \Psi(\pi_{Z'}(w))(b).\end{aligned}$$

Together with (9.10), this implies $\varphi'(\pi_{C'\cup P'}(w))(a) = \Psi(\pi_{Z'}(w))(a)$. Combining this with (9.9) and (9.10), we obtain $\varphi'(\pi_{C'\cup P'}(w)) = \Psi(\pi_{Z'}(w))$. This proves the counting property.

- *Commutative projection property.* Observe that

$$\begin{aligned}\Psi(\pi_{C'\cup P'}(K')) &= \Psi(\pi_{Y_1}(\hat{K})) = \Psi(\pi_{Y_1}(h^{-1}(K))) \\ &= \Psi(h^{-1}(\pi_{C\cup P}(K))) = \bigcup_{c\in C'} c + P_c'^{\oplus}.\end{aligned}$$

- *Boundedness.* Since $|w|_\diamond = |h(v)|_\diamond$ for each $w \in K'$ with $w = g(v)$, there is a constant bounding $|w|_\diamond$ for $w \in K'$.

- *Insertion property.* Let $cw \in K'$ with $c \in C'$ and $\mu \in P_c'^{\oplus}$. Then $f(\mu)$ is contained in $P_{f(c)}^{\oplus}$ and $f(cw)$ belongs to K. Write

$$\pi_{Z'\cup\{\diamond\}}(cw) = w_0 \diamond w_1 \diamond \cdots \diamond w_n$$

with $w_0, \ldots, w_n \in Z'^*$. Then

$$\pi_{Z\cup\{\diamond\}}(f(cw)) = f(\pi_{Z'\cup\{\diamond\}}(cw)) = f(w_0) \diamond \cdots \diamond f(w_n).$$

By the insertion property of K and since $f(cw) \in K$, there is a $v \in K$ with

$$\pi_Z(v) = f(w_0)v_1 f(w_1) \cdots v_n f(w_n),$$

$v_1, \ldots, v_n \in Z^*$, and $\Psi(\pi_Z(v)) = \Psi(\pi_Z(f(cw))) + \varphi(f(\mu))$. In particular, we have $\Psi(v_1 \cdots v_n) = \varphi(f(\mu))$. Note that $\varphi'(\mu) \in Z'^{\oplus}$ is obtained from $\varphi(f(\mu)) \in Z^{\oplus}$ by replacing some occurrences of a by b. Thus, by the definition of f, we can find words $v_1', \ldots, v_n' \in Z'^*$ with $f(v_i') = v_i$ and $\Psi(v_1' \cdots v_n') = \varphi'(\mu)$. Then the word

$$w' = w_0 v_1' w_1 \cdots v_n' w_n \in Z'^*$$

satisfies $\pi_{Z'\cup\{\diamond\}}(cw) \preceq_\diamond w'$, $\Psi(w') = \Psi(\pi_{Z'}(cw)) + \varphi'(\mu)$ and

$$f(w') = f(w_0)v_1 f(w_1) \cdots v_n f(w_n) = \pi_Z(v) \in \pi_Z(K) = L.$$

Since $f^{-1}(L) = \sigma(L)$, this means $w' \in \sigma(L)$. We have thus established the insertion property.

We conclude that the tuple $(K', C', P', (P'_c)_{c\in C'}, \varphi', \diamond)$ is a PAIM in G_i for $\sigma(L)$. □

We are now ready to describe the construction of PAIM for languages $\sigma(L)$ when given PAIM for L and for each $\sigma(x)$. Here, the result on letter substitutions (Lemma 9.2.15) allows us to assume that the PAIM for each $\sigma(x)$ is linear. However, it remains the obstacle to make sure that the number of occurrences of $\diamond$ stays bounded.

Lemma 9.2.16 (Substitutions)**.** *Let $L \subseteq X^*$ in G_i and σ be a G_i-substitution. Given a PAIM in G_i for L and for each $\sigma(x)$, $x \in X$, one can construct a PAIM for $\sigma(L)$ in G_i.*

Proof. Let $\sigma\colon X^* \to \mathcal{P}(Y^*)$. Assuming that for some $a \in X$, we have $\sigma(x) = \{x\}$ for all $x \in X \setminus \{a\}$ means no loss of generality. According to Lemma 9.2.8, we may also assume that $\sigma(a) \subseteq Z^*$ for some alphabet Z with $Y = X \uplus Z$. If $\sigma(a) = L_1 \cup \cdots \cup L_n$, then first substituting a by $\{a_1, \ldots, a_n\}$ and then each a_i by L_i has the same effect as applying σ. Hence, Lemma 9.2.15 allows us to assume further that the PAIM given for $\sigma(a)$ is linear. Finally, since we clearly have $\sigma(L) = (L \setminus X^*aX^*) \cup \sigma(L \cap X^*aX^*)$, Lemmas 9.2.7, 9.2.10, and 9.2.11 imply that we may also assume $L \subseteq X^*aX^*$.

Let $(K, C, P, (P_c)_{c\in C}, \varphi, \diamond)$ be a PAIM for L and $(\hat{K}, \hat{c}, \hat{P}, \hat{\varphi}, \diamond)$ be a linear PAIM for $\sigma(a)$. The idea of the construction is to replace each occurrence of a in K by words from $\hat{K}$ after removing $\hat{c}$. However, in order to guarantee a finite bound for the number of occurrences of $\diamond$ in the resulting words, we also remove $\diamond$ from all but one inserted words from $\hat{K}$. The new map φ' is then set up to so that if $f \in C \cup P$ represented m occurrences of a, then $\varphi'(f)$ will represent m times $\hat{\varphi}(\hat{c})$.

Let $C' = C$, $P'_c = P_c \cup \hat{P}$, $P' = \bigcup_{c\in C'} P'_c$, and $\varphi'\colon (C' \cup P')^{\oplus} \to Y^{\oplus}$ be the morphism with

$$\begin{aligned} \varphi'(f) &= \varphi(f) - \varphi(f)(a) \cdot a + \varphi(f)(a) \cdot \hat{\varphi}(\hat{c}) && \text{for } f \in C \cup P, \\ \varphi'(f) &= \hat{\varphi}(f) && \text{for } f \in \hat{P}. \end{aligned}$$

Let $a_\diamond$ be a new symbol and

$$\bar{K} = \{ua_\diamond v \mid uav \in K,\ |u|_a = 0\}.$$

In other words, $\bar{K}$ is obtained by replacing in each word from K the first occurrence of a with $a_\diamond$. The occurrence of $a_\diamond$ will be the one that is replaced by annotation words containing $\diamond$, whereas the occurrences of a are replaced by annotation words without $\diamond$. Hence, let τ be the substitution

$$\begin{aligned} \tau\colon (C \cup X \cup P \cup \{\diamond, a_\diamond\})^* &\longrightarrow \mathcal{P}((C' \cup Z \cup P' \cup \{\diamond\})^*) \\ x &\longmapsto \{x\}, \quad \text{for } x \in C \cup X \cup P \cup \{\diamond\},\ x \neq a, \\ a_\diamond &\longmapsto \pi_{Z \cup \hat{P} \cup \{\diamond\}}(\hat{K}), \\ a &\longmapsto \pi_{Z \cup \hat{P}}(\hat{K}). \end{aligned}$$

We claim that with $K' = \tau(\bar{K})$, the tuple $(K', C', P', (P'_c)_{c\in C'}, \varphi', \diamond)$ is a PAIM in G_i for $\sigma(L)$. First, since G_i is closed under rational transductions and substitutions, K' is in G_i.

- *Projection property.* Since $L = \pi_X(K)$ and $\sigma(a) = \pi_Z(\hat{K})$, we have in fact the equality $\sigma(L) = \pi_Z(K')$.
- *Counting property.* Let $w \in K'$. Then by the definition of K', there is a word $u = cu_0au_1 \cdots au_n \in K$ with $u_i \in (C \cup X \cup P \cup \{\diamond\})^*$, $c \in C$, and $|u_i|_a = 0$ for $i = 0, \ldots, n$ such that $w = cu_0w_1u_1 \cdots w_nu_n$ with $w_1 \in \pi_{Z \cup \hat{P} \cup \{\diamond\}}(\hat{K})$ and $w_i \in \pi_{Z \cup \hat{P}}(\hat{K})$ for $i = 2, \ldots, n$. The counting property of $\hat{K}$ yields

$$\Psi(\pi_Z(w_i)) = \hat{\varphi}(\hat{c}) + \hat{\varphi}(\pi_{\hat{P}}(w_i)). \tag{9.11}$$

  Since $\varphi(\pi_{C \cup P}(u))(a) = \Psi(\pi_X(u)) = n$, we have

$$\begin{aligned} \varphi'(\pi_{C' \cup P'}(u)) &= \varphi(\pi_{C \cup P}(u)) - n \cdot a + n \cdot \hat{\varphi}(\hat{c}) \\ &= \Psi(\pi_X(u)) - n \cdot a + n \cdot \hat{\varphi}(\hat{c}). \end{aligned} \tag{9.12}$$

  Equations (9.11) and (9.12) together imply

$$\begin{aligned} \varphi'(\pi_{C' \cup P'}(w)) &= \varphi'(\pi_{C' \cup P'}(u)) + \sum_{i=1}^{n} \varphi'(\pi_{P'}(w_i)) \\ &= \Psi(\pi_X(u)) - n \cdot a + n \cdot \hat{\varphi}(\hat{c}) + \sum_{i=1}^{n} (\Psi(\pi_Z(w_i)) - \hat{\varphi}(\hat{c})) \\ &= \Psi(\pi_X(u)) - n \cdot a + \sum_{i=1}^{n} \Psi(\pi_Z(w_i)) = \Psi(\pi_Z(w)). \end{aligned}$$

- *Commutative projection property.* Let $c \in C'$ and $\mu \in P_c'^{\oplus}$. By definition of P_c', we can write $\mu = \nu + \hat{\nu}$ with $\nu \in P_c^{\oplus}$ and $\hat{\nu} \in \hat{P}^{\oplus}$. By the commutative projection property of K, we find a $cw \in K$ with $\Psi(\pi_{C \cup P}(cw)) = c + \nu$. Since $L \subseteq X^*aX^*$, we can write $w = cu_0au_1 \cdots au_n$ with $|u_i|_a = 0$ for $0 \leqslant i \leqslant n$ and $n \geqslant 1$. Moreover, the commutative projection property of $\hat{K}$ yields words $\hat{c}\hat{w} \in \hat{K}$ and $\hat{c}\hat{w}' \in \hat{K}$ with $\Psi(\pi_{\hat{c} \cup \hat{P}}(\hat{c}\hat{w})) = \hat{c} + \hat{\nu}$ and $\Psi(\pi_{\hat{c} \cup \hat{P}}(\hat{c}\hat{w}')) = \hat{c}$. By definition of K', the word

$$w' = cu_0\hat{w}u_1\hat{w}'u_2 \cdots \hat{w}'u_n$$

  is in K' and satisfies $\Psi(\pi_{C' \cup P'}(w')) = c + \nu + \hat{\nu} = c + \mu$. This proves

$$\bigcup_{c \in C'} c + P_c'^{\oplus} \subseteq \Psi(\pi_{C' \cup P'}(K')).$$

  The other inclusion is clear by definition. We have thus established that the tuple $(\pi_{C' \cup Z \cup P'}(K'), C', P', (P_c')_{c \in C'}, \varphi')$ is a Parikh annotation in G_i for $\sigma(L)$.
- *Boundedness.* Note that if $|w|_\diamond \leqslant k$ for all $w \in K$ and $|\hat{w}|_\diamond \leqslant \ell$ for all $\hat{w} \in \hat{K}$, then $|w'|_\diamond \leqslant k + \ell$ for all $w' \in K'$ by construction of K', implying boundedness.
- *Insertion property.* The insertion property follows from the insertion property of K and $\hat{K}$.

□

9.2.3 One nonterminal

The next step is to construct PAIM for languages $L(G)$, where G has just one nonterminal S and PAIM are given for the right-hand-sides. This essentially amounts to constructing a PAIM for $SF(G)$ in the case that S occurs in every word on the right-hand side (Lemma 9.2.17), because then, $L(G)$ can be obtained from $SF(G)$ using a substitution. As in the proof of Lemma 2.6.10, we observe that applying $S \to L$ for some $w \in L$ contributes $\Psi(w) - S$ to the Parikh image of the sentential form. This means $\Psi(SF(G)) = S + (\Psi(L) - S)^{\oplus}$. Therefore, computing a PAIM for $SF(G)$ is akin to computing a semilinear representation for $U^{\oplus}$, where U is a semilinear set.

Lemma 9.2.17 (Sentential forms)**.** *Let* $G = (N, T, P, S)$ *be an* G_i*-grammar with* $N = \{S\}$, $P = \{S \to L\}$, *and* $L \subseteq (N \cup T)^* S (N \cup T)^*$. *Furthermore, suppose a PAIM in* G_i *is given for* L. *Then one can construct a PAIM in* G_i *for* $SF(G)$.

Proof. As explained above, for the construction of the PAIM, we use an idea to obtain a semilinear representation of $U^{\oplus}$ for semilinear sets U. If $U = \bigcup_{j=1}^{n} \mu_j + F_j^{\oplus}$ for $\mu_j \in X^{\oplus}$ and finite $F_j \subseteq X^{\oplus}$, then

$$U^{\oplus} = \bigcup_{D \subseteq \{1,\dots,n\}} \left(\sum_{j \in D} \mu_j \right) + \left(\bigcup_{j \in D} \{\mu_j\} \cup F_j \right)^{\oplus}.$$

The symbols representing constant and period vectors for $SF(G)$ are therefore set up as follows. Let $(K, C, P, (P_c)_{c \in C}, \varphi, \diamond)$ be a PAIM for L in G_i. We define S' and S_D and d_D to be new symbols for each $D \subseteq C$. Moreover, we use the sets $C' = \{d_D \mid D \subseteq C\}$ and $P' = C \cup P$ with $P'_{d_D} = D \cup \bigcup_{c \in D} P_c$. We will use the shorthand $X = N \cup T$. Observe that since $L \subseteq X^* S X^*$, we have $\varphi(c)(S) \geqslant 1$ for each $c \in C$. We can therefore define the morphism $\varphi' \colon (C' \cup P')^{\oplus} \to X^{\oplus}$ as

$$\begin{aligned} \varphi'(p) &= \varphi(p) && \text{for } p \in P, \\ \varphi'(c) &= \varphi(c) - S && \text{for } c \in C, \qquad (9.13) \\ \varphi'(d_D) &= S + \sum_{c \in D} \varphi'(c). && \qquad (9.14) \end{aligned}$$

The essential idea in our construction is to use modified versions of K as right-hand-sides of a grammar. These modified versions are obtained as follows. For each $D \subseteq C$, we define the rational transduction δ_D which maps each word $w_0 S w_1 \cdots S w_n \in (C \cup X \cup P \cup \{\diamond\})^*$, $|w_i|_S = 0$ for $0 \leqslant i \leqslant n$, to all words $w_0 S_{D_1} w_1 \cdots S_{D_n} w_n$ for which

$$D_1 \cup \cdots \cup D_n = D, \qquad D_i \cap D_j = \emptyset \text{ for } i \neq j.$$

Thus, δ_D can be thought of as distributing the elements of D among the occurrences of S in the input word. The modified versions of K are then given by

$$K_D = \delta_D(\pi_{C \cup X \cup P}(K)), \qquad K_D^c = \delta_{D \setminus c}(c^{-1} K).$$

In the new annotation, the symbol d_D represents $S + \sum_{c \in D} (\varphi(c) - S)$. Since each symbol $c \in C$ still represents $\varphi(c) - S$, we cannot insert a whole word from

K for each inserted word from L: This would insert a $c \in C$ in each step and we would count $\sum_{c \in D}(\varphi(c) - S)$ twice. Hence, in order to compensate for the new constant symbol d_D, when generating a word starting with d_D, we have to prevent exactly one occurrence of c for each $c \in D$ from appearing. To this end, we use the nonterminal S_D, which only allows derivation subtrees in which of each $c \in D$, precisely one occurrence has been left out, i.e. a production $S_D \to K_D^c$ (for some $D \subseteq C$) has been applied. In the productions $S_D \to K_D$ the symbol from C on the right-hand side is allowed to appear.

In order to have only a bounded number of occurrences of $\diamond$, one of our modified versions of K (namely K_D^c) introduces $\diamond$ and the other one (K_D) does not. Since when generating a word starting with d_D, our grammar makes sure that for each $c \in D$, a production of the form $S_E \to K_E^c$ is used precisely once (and otherwise $S_E \to K_E$), the set K_E^c is set up to contain $\diamond$. This will guarantee that during the insertion process simulating $S \to L$, we insert at most $|C| \cdot \ell$ occurrences of $\diamond$, where ℓ is an upper bound for $|w|_\diamond$ for $w \in K$.

Let $N' = \{S'\} \cup \{S_D \mid D \subseteq C\}$ and let $\hat{P}$ consist of the following productions:

$$S' \to \{d_D \diamond S_D \diamond \mid D \subseteq C\} \tag{9.15}$$

$$S_\emptyset \to \{S\} \tag{9.16}$$

$$S_D \to K_D \qquad \text{for each } D \subseteq C \tag{9.17}$$

$$S_D \to K_D^c \qquad \text{for each } D \subseteq C \text{ and } c \in D. \tag{9.18}$$

Finally, let M be the regular language

$$M = \bigcup_{D \subseteq C} \{w \in (C' \cup X \cup P' \cup \{\diamond\})^* \mid \pi_{C' \cup P'}(w) \in d_D P'^*_{d_D}\}.$$

By intersecting with M, we make sure that the commutative projection property is satisfied. We shall prove that with $G' = (N', C' \cup X \cup P' \cup \{\diamond\}, \hat{P}, S')$ and the language $K' = L(G') \cap M$, the tuple $(K', C', P', (P'_c)_{c \in C'}, \varphi', \diamond)$ is a PAIM for $\mathsf{SF}(G)$ in G_i. By definition, $L(G')$ is contained in $\mathsf{Alg}(G_i) = G_i$ and hence K' since G_i is a full semi-AFL.

Let $h\colon (N' \cup C' \cup X \cup P' \cup \{\diamond\})^* \to (C' \cup X \cup P' \cup \{\diamond\})^*$ be the morphism that fixes $C' \cup X \cup P' \cup \{\diamond\}$ and satisfies $h(S') = h(S_D) = S$ for $D \subseteq C$. Moreover, regard $\mathcal{P}(C)$ as a monoid with $\cup$ as its operation. Then $\rho\colon (N' \cup X)^* \to \mathcal{P}(C)$ is the morphism with $\rho(S_D) = D$ and $\rho(S') = \rho(x) = \emptyset$ for $x \in X$. Furthermore, let $|w|_\diamond \leqslant \ell$ for all $w \in K$. We claim that for each $n \geqslant 0$, $d_D \diamond S_D \diamond \Rightarrow_{G'}^n w$ implies

1. if $w = u_0 S_{D_1} u_1 \cdots S_{D_n} u_n$ with $u_i \in X^*$ for $0 \leqslant i \leqslant n$, then $D_i \cap D_j = \emptyset$ for $i \neq j$.
2. $h(\pi_{N' \cup X}(w)) \in \mathsf{SF}(G)$,
3. $\varphi'(\pi_{C' \cup P'}(w)) = \Psi(h(\pi_{N' \cup X}(w))) + \sum_{c \in \rho(w)} \varphi'(c)$,
4. $|w|_\diamond \leqslant 2 + |D \setminus \rho(w)| \cdot \ell$, and
5. for each $\mu \in \left(D \cup \bigcup_{c \in D \setminus \rho(w)} P_c\right)^\oplus$, there is a $w' \in \mathsf{SF}(G)$ such that $h(\pi_{N' \cup X \cup \{\diamond\}}(w)) \preceq_\diamond w'$ and $\Psi(w') = \Psi(h(\pi_{N' \cup X}(w))) + \varphi'(\mu)$.

We establish this claim using induction on n. Observe that all these conditions are satisfied in the case $n = 0$, i.e. $w = d_D \diamond S_D \diamond$, conditions 1 to 4 follow directly by distinguishing among the productions in G'. Therefore, we only prove condition 5 in the induction step.

Suppose $n > 0$ and $d_D \diamond S_D \diamond \Rightarrow_{G'}^{n-1} \bar{w} \Rightarrow_{G'} w$. If the production applied in $\bar{w} \Rightarrow_{G'} w$ is $S_\emptyset \to \{S\}$, then $\rho(w) = \rho(\bar{w})$ and

$$h(\pi_{N' \cup X \cup \{\diamond\}})(w) = h(\pi_{N' \cup X \cup \{\diamond\}})(\bar{w}),$$

so that condition 5 follows immediately from the same condition for $\bar{w}$. If the applied production is of the form (9.17) or (9.18), then we have $\rho(w) \subseteq \rho(\bar{w})$ and hence $\rho(\bar{w}) = \rho(w) \cup E$ for some $E \subseteq D$, $|E| \leqslant 1$. Then

$$\bigcup_{c \in D \setminus \rho(w)} P_c = \bigcup_{c \in D \setminus \rho(\bar{w})} P_c \cup \bigcup_{c \in E} P_c.$$

We can therefore decompose $\mu \in \left(D \cup \bigcup_{c \in D \setminus \rho(w)} P_c\right)^\oplus$ into $\mu = \bar{\mu} + \nu$ with $\bar{\mu} \in \left(D \cup \bigcup_{c \in D \setminus \rho(\bar{w})} P_c\right)^\oplus$ and $\nu \in \left(\bigcup_{c \in E} P_c\right)^\oplus$. By induction, we find a $\bar{w}' \in SF(G)$ such that

$$h(\pi_{N' \cup X \cup \{\diamond\}}(\bar{w})) \preceq_\diamond \bar{w}', \quad \Psi(\bar{w}') = \Psi(h(\pi_{N' \cup X}(\bar{w}))) + \varphi'(\bar{\mu}).$$

Let $\bar{w} = xSy$ be the decomposition facilitating the step $\bar{w} \Rightarrow_{G'} w$ and define $w = xzy$.

- Suppose the production applied in $\bar{w} \Rightarrow_{G'} w$ is of the form (9.17). Then $\rho(w) = \rho(\bar{w})$ and hence $E = \emptyset$ and $\nu = 0$. Furthermore, $z \in K_F$ for some $F \subseteq C$. We define $z' = h(\pi_{N' \cup X}(z))$. Note that then $z' \in \pi_X(K) = L$ and $\Psi(z') = \Psi(\pi_X(z)) + \varphi'(\nu)$.
- Suppose the production applied in $\bar{w} \Rightarrow_{G'} w$ is of the form (9.18). Then $z \in K_F^c$ for some $c \in F \subseteq C$ and thus $h(z) \in c^{-1}K$. This implies that $\rho(\bar{w}) = \rho(w) \cup \{c\}$, $E = \{c\}$, and hence $\nu \in P_c^\oplus$. The insertion property of K provides a $z' \in L$ such that $\pi_{X \cup \{\diamond\}}(h(z)) \preceq_\diamond z'$ and

$$\Psi(z') = \Psi(\pi_X(h(z))) + \varphi(\nu) = \Psi(\pi_X(h(z))) + \varphi'(\nu).$$

In any case, we have

$$z' \in L, \qquad h(\pi_{N' \cup X \cup \{\diamond\}}(z)) \preceq_\diamond z', \qquad \Psi(z') = \Psi(h(\pi_{N' \cup X}(z))) + \varphi'(\nu).$$

Recall that $\bar{w} = xSy$ and $w = xzy$. Since $\bar{w} \preceq_\diamond \bar{w}'$, we can find x', y' with

$$\bar{w}' = x'Sy', \qquad h(\pi_{N' \cup X}(x)) \preceq_\diamond x', \qquad h(\pi_{N' \cup X}(y)) \preceq_\diamond y'.$$

Choose $w' = x'z'y'$. Then $SF(G) \ni \bar{w}' \Rightarrow_G w'$ and thus $w' \in SF(G)$. Moreover,

$$\begin{aligned} h(\pi_{N' \cup X \cup \{\diamond\}}(w)) &= h(\pi_{N' \cup X \cup \{\diamond\}}(x)) h(\pi_{N' \cup X \cup \{\diamond\}}(z)) h(\pi_{N' \cup X \cup \{\diamond\}}(y)) \\ &\preceq_\diamond x'z'y' = w'. \end{aligned}$$

Finally, w' has the desired Parikh image:

$$\begin{aligned}\Psi(w') &= \Psi(\bar{w}') - S + \Psi(z') \\ &= \Psi(h(\pi_{N'\cup X}(\bar{w}))) + \varphi'(\bar{\mu}) - S + \Psi(z') \\ &= \Psi(h(\pi_{N'\cup X}(\bar{w}))) + \varphi'(\bar{\mu}) - S + \Psi(h(\pi_{N'\cup X}(z))) + \varphi'(\nu) \\ &= \Psi(h(\pi_{N'\cup X}(w))) + \varphi'(\bar{\mu}) + \varphi'(\nu) \\ &= \Psi(h(\pi_{N'\cup X}(w))) + \varphi'(\mu).\end{aligned}$$

This completes the induction step for condition 5.

We now use our claim to prove that we have indeed constructed a PAIM.

- *Projection property.* Our claim already entails $\pi_X(K') \subseteq SF(G)$: For words $w \in (C' \cup X \cup P' \cup \{\diamond\})^*$ with $d_D \diamond S_D \diamond \Rightarrow_{G'}^* w$, the projection $\pi_X(w)$ equals $h(\pi_{N'\cup X}(w))$, which belongs to $SF(G)$ by condition 2. In order to prove $SF(G) \subseteq \pi_X(K')$, suppose $w \in SF(G)$ and let t be a partial derivation tree for G with root label S and $\mathsf{yield}(t) = w$. Since $c(x) \in L$ for each inner node x of t, we can find a $c_x w_x \in K$ with $\pi_X(c_x w_x) = c(x)$. Then in particular $c(x) \preceq c_x w_x$, meaning we can obtain a tree t' from t as follows: For each inner node x of t, add new leaves directly below x so as to have $c_x w_x$ as the new sequence of child labels of x. Note that the set of inner nodes of t' is identical to the one of t. Moreover, we have $\pi_X(\mathsf{yield}(t')) = w$.

  Let $D = \{c_x \mid x \text{ is an inner node in } t'\}$. We pick for each $c \in D$ exactly one inner node x in t' such that $c_x = c$; we denote the resulting set of nodes by R. We now obtain t'' from t' as follows: For each $x \in R$, we remove its c_x-labeled child; for each $x \notin R$, we remove all $\diamond$-labeled children. Note that again, the inner nodes of t'' are the same as in t and t'. Moreover, we still have $\pi_X(\mathsf{yield}(t'')) = w$.

  For each inner node x in t'', let $D_x = \{c_y \mid y \in R \text{ is below } x \text{ in } t''\}$. Note that in t, t', t'', every inner node has the label S. We obtain the tree t''' from t'' as follows. For each inner node x in t'', we replace its label S by S_{D_x}. Then we have $\pi_X(h(\mathsf{yield}(t''')) = w$. Clearly, the root node of t''' is labeled S_D. Furthermore, the definition of K_E and K_E^c yields that t''' is a partial derivation tree for G'. Hence

  $$S' \;\Rightarrow_{G'}\; d_D \diamond S_D \diamond \;\Rightarrow_{G'}^*\; d_D \diamond \mathsf{yield}(t''') \diamond.$$

  Since in t''', every leaf has a label in $T \cup \{S_\emptyset\}$, we have

  $$S' \Rightarrow_{G'}^* d_D \diamond h(\mathsf{yield}(t'''))\diamond,$$

  and hence $d_D \diamond h(\mathsf{yield}(t'''))\diamond \in L(G')$. The word $d_D \diamond h(\mathsf{yield}(t'''))\diamond$ is also clearly contained in M and since $\pi_X(d_D \diamond h(\mathsf{yield}(t'''))\diamond) = w$, this implies $w \in \pi_X(K')$.

- *Counting property.* Suppose $w \in (C' \cup X \cup P' \cup \{\diamond\})^*$ with $d_D \diamond S_D \diamond \Rightarrow_{G'}^* w$. We apply condition 3 in our claim to w. Since we have $\rho(w) = \emptyset$ and also $h(\pi_{N'\cup X}(w)) = \pi_X(w)$, this yields $\varphi'(\pi_{C'\cup P'}(w)) = \Psi(\pi_X(w))$. In particular, this equality holds for elements of K'.

- *Commutative projection property.* Since we have $K' \subseteq M$, there is clearly an inclusion $\Psi(\pi_{C'\cup P'}(K')) \subseteq \bigcup_{c\in C'} c + P_c'^{\oplus}$.

For the other inclusion, let $D \subseteq C$ with $D = \{c_1, \ldots, c_n\}$. Suppose that $\mu \in \bigcup_{c \in C'} c + P'^{\oplus}_c$, $\mu = d_D + \nu + \sum_{i=1}^{n} \xi_i$ with $\nu \in D^{\oplus}$ and $\xi_i \in P^{\oplus}_{c_i}$ for $1 \leqslant i \leqslant n$.

The commutative projection property of K allows us to choose for each index $1 \leqslant i \leqslant n$ words $u_i, v_i \in K$ such that

$$\Psi(\pi_{C \cup P}(u_i)) = c_i, \qquad \Psi(\pi_{C \cup P}(v_i)) = c_i + \xi_i.$$

The words $v'_0, \ldots, v'_n$ are constructed as follows. Let $v'_0 = d_D \diamond S \diamond$ and let v'_{i+1} be obtained from v'_i by replacing the first occurrence of S by $c_{i+1}^{-1} v_{i+1}$. Furthermore, let v''_i be obtained from v'_i by replacing the first occurrence of S by $S_{\{c_{i+1}, \ldots, c_n\}}$ and all other occurrences by $S_\emptyset$. Then we clearly have the derivation

$$d_D \diamond S_D \diamond = v''_0 \Rightarrow_{G'} \cdots \Rightarrow_{G'} v''_n$$

and $v''_n \in (T \cup \{S_\emptyset\})^*$. Moreover, we have $\Psi(\pi_{C' \cup P'}(v''_n)) = d_D + \sum_{i=1}^{n} \xi_i$.

Let $g \colon X^* \to (T \cup \{S_\emptyset\})^*$ be the morphism with $g(S) = S_\emptyset$ and that fixes the elements of T. For a word $w \in (N' \cup X)^*$ that contains $S_\emptyset$ and for $1 \leqslant i \leqslant n$, let $U_i(w)$ be the word obtained from w by replacing the first occurrence of $S_\emptyset$ by $g(u_i)$. Then $U_i(w)$ satisfies $w \Rightarrow_{G'} U_i(w)$ and the equality $\Psi(\pi_{C' \cup P'}(U_i(w))) = \Psi(\pi_{C' \cup P'}(w)) + c_i$. Thus, with

$$u = U_n^{\nu(c_n)} \cdots U_1^{\nu(c_1)}(v''_n)$$

(that is, we apply $\nu(c_1)$ times the function U_1, then $\nu(c_2)$ times the function U_2, etc.), we have $v''_n \Rightarrow^*_{G'} u \Rightarrow^*_{G'} h(u)$ and hence $h(u) \in L(G')$. By construction, $h(u)$ belongs to M and thus $h(u) \in K'$. Moreover, we have

$$\begin{aligned} \Psi(\pi_{C' \cup P'}(h(u))) &= \Psi(\pi_{C' \cup P'}(u)) = \Psi(\pi_{C' \cup P'}(v''_n)) + \nu \\ &= d_D + \sum_{i=1}^{n} \xi_i + \nu = \mu. \end{aligned}$$

This proves $\bigcup_{c \in C'} c + P'^{\oplus}_c \subseteq \Psi(\pi_{C' \cup P'}(K'))$.

- *Boundedness.* Let $w \in (C' \cup X \cup P' \cup \{\diamond\})^*$ and $d_D \diamond S_D \diamond \Rightarrow^*_{G'} w$. By condition 4 of our claim, we have $|w|_\diamond \leqslant 2 + |C| \cdot \ell$.
- *Insertion property.* Let $w \in (C' \cup X \cup P' \cup \{\diamond\})^*$ and $d_D \diamond S_D \diamond \Rightarrow^*_{G'} w$. Then $\rho(w) = \emptyset$ and $h(\pi_{N' \cup X \cup \{\diamond\}}(w)) = \pi_{X \cup \{\diamond\}}(w)$. Hence condition 5 states that for each $\mu \in P'^{\oplus}_{d_D}$, there is a $w' \in SF(G)$ with $\pi_{X \cup \{\diamond\}}(w) \preceq_\diamond w'$ and $\Psi(w') = \Psi(\pi_X(w)) + \varphi'(\mu)$.

□

Lemma 9.2.18 (One nonterminal)**.** *Let G be a G_i-grammar with one nonterminal. Furthermore, suppose PAIM in G_i are given for the right-hand-sides in G. Then we can construct a PAIM for $L(G)$ in G_i.*

Proof. Let $G = (N, T, P, S)$. By Lemma 9.2.7, we may assume that there is only one production $S \to L$ in P. By Lemmas 9.2.10 and 9.2.11, one can construct PAIM for $L_0 = L \setminus (N \cup T)^* S (N \cup T)^*$ and for $L_1 = L \cap (N \cup T)^* S (N \cup T)^*$.

We define G' to be the grammar $G' = (N, T, P', S)$, in which $P' = \{S \to L_1\}$. Moreover, let $\sigma\colon (N \cup T)^* \to \mathcal{P}((N \cup T)^*)$ be the substitution with $\sigma(S) = L_0$ and $\sigma(t) = \{t\}$ for $t \in T$. Then we have $L(G) = \sigma(\mathsf{SF}(G'))$. Hence, one can construct a PAIM for $L(G)$ using Lemmas 9.2.16 and 9.2.17. □

Thanks to Lemmas 9.2.16 and 9.2.18, we can now use the van Leeuwen decomposition (see Section 2.6, Page 32) to construct PAIM recursively with respect to the number of nonterminals in G.

Lemma 9.2.19 (PAIM for algebraic extensions)**.** *Given $i \in \mathbb{N}$ and an F_i-grammar G, along with a PAIM in F_i for each right-hand side, one can construct a PAIM for $L(G)$ in G_i.*

Proof. Our algorithm works recursively with respect to the number of nonterminals. In order to make the recursion work, we need the algorithm to work with right-hand sides in G_i. We show that, given $i \in \mathbb{N}$, a G_i-grammar G, along with a PAIM in G_i for each right-hand side in G, we can construct a PAIM for $L(G)$ in G_i. Since a PAIM for a language L in F_i can easily be turned into a PAIM for L in G_i, this statement implies the lemma.

Let $G = (N, T, P, S)$ be a G_i-grammar with $n = |N|$ and let G_A and G' be a van Leeuwen decomposition of G. Since G_A has only one nonterminal and its right-hand sides are among those of G, we have a PAIM for each right-hand side of G_A and Lemma 9.2.18 allows us to construct a PAIM for $L(G_A)$. This means, the right-hand sides of G' are obtained by substitutions from languages for which we have PAIMs. Hence, we can use Lemma 9.2.16 to construct a PAIM in G_i for each right-hand side of G'. Since G' has $n - 1$ nonterminals, we can recursively construct a PAIM for $L(G') = L(G)$. □

The last step is to compute PAIM for languages in $\mathsf{SLI}(G_i)$.

Lemma 9.2.20 (Semilinear intersection)**.** *Given $i \in \mathbb{N}$, a language $L \subseteq X^*$ in G_i, a semilinear $S \subseteq X^\oplus$, and a morphism $h\colon X^* \to Y^*$, along with a PAIM in G_i for L, one can construct a PAIM for $h(L \cap \Psi^{-1}(S))$ in $\mathsf{SLI}(G_i)$.*

Proof. According to Lemma 9.2.8, it suffices to show that we can construct a PAIM for $L \cap \Psi^{-1}(S)$. Moreover, if $L = L_1 \cup \cdots L_n$, then

$$L \cap \Psi^{-1}(S) = (L_1 \cap \Psi^{-1}(S)) \cup \cdots \cup (L_n \cap \Psi^{-1}(S)).$$

Thus, by Lemmas 9.2.7 and 9.2.9, we may assume that the PAIM for L is linear. Let $(K, c, P, \varphi, \diamond)$ be a linear PAIM for L in G_i.

The set $T = \{\mu \in P^\oplus \mid \varphi(c + \mu) \in S\}$ is semilinear as well, hence $T = \bigcup_{i=1}^n T_i$ for linear $T_i \subseteq P^\oplus$. Write $T_i = \mu_i + F_i^\oplus$ with $\mu_i \in P^\oplus$, and $F_i \subseteq P^\oplus$ being a finite set. Let P_i' be an alphabet with new symbols in bijection with the set F_i and let $\psi_i\colon P_i'^\oplus \to P^\oplus$ be the morphism extending this bijection. Moreover, let U_i be the linear set

$$U_i = \mu_i + \{p + \psi_i(p) \mid p \in P_i'\}^\oplus + (X \cup \{\diamond\})^\oplus$$

and let $R_i = p_1^* \cdots p_m^*$, where $P_i' = \{p_1, \ldots, p_m\}$. We claim that with new symbols

c'_i for $1 \leqslant i \leqslant n$, $C' = \{c'_i \mid 1 \leqslant i \leqslant n\}$, $P' = \bigcup_{i=1}^{n} P'_i$ and

$$\begin{aligned} \varphi'(c'_i) &= \varphi(c) + \varphi(\mu_i), \\ \varphi'(p) &= \varphi(\psi_i(p)) \qquad \text{for } p \in P'_i, \\ K' &= \bigcup_{i=1}^{n} c'_i \pi_{C' \cup X \cup P' \cup \{\diamond\}} \left(c^{-1} K R_i \cap \Psi^{-1}(U_i) \right), \end{aligned}$$

the tuple $(K', C', P', (P'_i)_{c'_i \in C'}, \varphi', \diamond)$ is a PAIM for $L \cap \Psi^{-1}(S)$.

- *Projection property.* For $w \in L \cap \Psi^{-1}(S)$, we find a $cv \in K$ that satisfies $\pi_X(cv) = w$. Then $\varphi(\pi_{\{c\} \cup P}(cv)) = \Psi(w) \in S$ and hence $\Psi(\pi_P(v)) \in T$. By definition of the P'_i, there is a decomposition $\Psi(\pi_P(v)) = \mu_i + \psi_i(\kappa)$ for some $1 \leqslant i \leqslant n$ and some $\kappa \in P_i'^{\oplus}$. Let $P'_i = \{p_1, \ldots, p_m\}$. Then the word

  $$v' = v p_1^{\kappa(p_1)} \cdots p_m^{\kappa(p_m)}$$

  is in $c^{-1} K R_i \cap \Psi^{-1}(U_i)$ and satisfies $\pi_X(v') = \pi_X(v) = w$. Moreover, $v'' = c'_i \pi_{C' \cup X \cup P' \cup \{\diamond\}}(v') \in K'$ and hence $w = \pi_X(v'') \in \pi_X(K')$. This proves $L \cap \Psi^{-1}(S) \subseteq \pi_X(K')$.

  We clearly have $\pi_X(K') \subseteq \pi_X(K) = L$. It therefore suffices to prove the inclusion $\Psi(\pi_X(K')) \subseteq S$. Suppose $w = c'_i v \in K'$. Then we can find some $v' \in c^{-1} K R_i \cap \Psi^{-1}(U_i)$ such that we have $v = \pi_{C' \cup X \cup P' \cup \{\diamond\}}(v')$. Let $P'_i = \{p_1, \ldots, p_m\}$. Then we can write $v' = v'' p_1^{\kappa(p_1)} \cdots p_m^{\kappa(p_m)}$ for some $\kappa \in P_i'^{\oplus}$. This means $cv'' \in K$ and thus $\Psi(\pi_X(cv'')) = \varphi(\pi_{\{c\} \cup P}(cv''))$ by the counting property of K. Since $v' \in \Psi^{-1}(U_i)$, we have

  $$\Psi(\pi_P(cv'')) = \Psi(\pi_P(v')) = \mu_i + \psi_i(\kappa) \in T_i.$$

  This implies

  $$\begin{aligned} \Psi(\pi_X(w)) &= \Psi(\pi_X(c'_i v)) = \Psi(\pi_X(v')) = \Psi(\pi_X(cv'')) \\ &= \varphi(\pi_{\{c\} \cup P}(cv'')) \in \varphi(c + T_i) \in S. \end{aligned}$$

- *Counting property.* Let $w = c'_i v \in K'$ with $v = \pi_{C' \cup X \cup P' \cup \{\diamond\}}(v')$ for some $v' \in c^{-1} K R_i \cap \Psi^{-1}(U_i)$. By definition of U_i, this implies

  $$\pi_P(v') = \mu_i + \psi_i(\pi_{P'}(v'))$$

  and hence

  $$\varphi(\pi_P(v')) = \varphi(\mu_i) + \varphi(\psi_i(\pi_{P'}(v'))) = \varphi(\mu_i) + \varphi'(\pi_{P'}(v')).$$

  Moreover, if we write $v' = v''r$ with $cv'' \in K$ and $r \in R_i$, then

  $$\begin{aligned} \varphi'(\pi_{C' \cup P'}(w)) &= \varphi'(c'_i) + \varphi'(\pi_{P'}(v')) \\ &= \varphi(c) + \varphi(\mu_i) + \varphi'(\pi_{P'}(v')) \\ &= \varphi(c) + \varphi(\pi_P(v')) = \varphi(\pi_{C \cup P}(cv'')) \\ &= \Psi(\pi_X(cv'')) = \Psi(\pi_X(w)). \end{aligned}$$

  This proves the counting property.

- *Commutative projection property.* Let $\mu \in c_i' + P_i'^{\oplus}$. This means we have $\mu = c_i' + \kappa$ with some $\kappa \in P_i'^{\oplus}$. Then $\xi = \psi_i(\kappa)$ belongs to $P^{\oplus}$ and the commutative projection property of K yields a $cv \in K$ for which we have $\Psi(\pi_{C\cup P}(cv)) = c + \mu_i + \xi$. Let $P_i' = \{p_1, \ldots, p_m\}$. Then the word

$$v' = v p_1^{\kappa(p_1)} \cdots p_m^{\kappa(p_m)}$$

  is contained in $c^{-1}KR_i \cap \Psi(U_i)$. Furthermore, $\Psi(\pi_{P'}(v')) = \kappa$ and hence

$$\Psi(\pi_{C'\cup P'}(c_i'\pi_{C'\cup X\cup P'\cup\{\diamond\}}(v'))) = c_i' + \kappa = \mu.$$

  This proves $\bigcup_{i=1}^{n} c_i' + P_i'^{\oplus} \subseteq \Psi(\pi_{C'\cup P'}(K'))$. The other inclusion follows directly from the definition of K'.

- *Boundedness.* Since $\pi_{\{\diamond\}}(K') \subseteq \pi_{\{\diamond\}}(K)$, K' inherits boundedness from K.

- *Insertion property.* Let $c_i'w \in K'$ and $\mu \in P_i'^{\oplus}$. Write $w = \pi_{C'\cup X\cup P'\cup\{\diamond\}}(v)$ for some $v \in c^{-1}KR_i \cap \Psi^{-1}(U_i)$, and $v = v'r$ for some $r \in R_i$. Then cv' belongs to K and applying the insertion property of K to cv' and to $\psi_i(\mu) \in P^{\oplus}$ yields a $v'' \in L$ with $\pi_{X\cup\{\diamond\}}(cv') \preceq_\diamond v''$ and

$$\Psi(v'') = \Psi(\pi_X(cv')) + \varphi(\psi_i(\mu)).$$

  This word satisfies

$$\begin{aligned}\pi_{X\cup\{\diamond\}}(c_i'w) &= \pi_{X\cup\{\diamond\}}(v) = \pi_{X\cup\{\diamond\}}(cv') \preceq_\diamond v'',\\ \Psi(\pi_X(v'')) &= \Psi(\pi_X(cv')) + \varphi(\psi_i(\mu))\\ &= \Psi(\pi_X(c_i'w)) + \varphi(\psi_i(\mu)) = \Psi(\pi_X(c_i'w)) + \varphi'(\mu).\end{aligned}$$

  and it remains to be shown that $v'' \in L \cap \Psi^{-1}(S)$. Since $v'' \in L$, this amounts to showing $\Psi(v'') \in S$.

  Since $\Psi(v') \in U_i$, we have $\Psi(\pi_P(v')) \in \mu_i + F_i^{\oplus}$ and $\psi_i(\mu) \in F_i^{\oplus}$ and hence also $\Psi(\pi_P(v')) + \psi_i(\mu) \in \mu_i + F_i^{\oplus} = T_i$. Therefore,

$$\begin{aligned}\Psi(v'') &= \Psi(\pi_X(cv')) + \varphi(\psi_i(\mu))\\ &= \varphi(\pi_{C\cup P}(cv')) + \varphi(\psi_i(\mu))\\ &= \varphi(\pi_{C\cup P}(cv') + \psi_i(\mu)) \in \varphi(c + T_i) \subseteq S.\end{aligned}$$

□

The foregoing lemmas now almost immediately imply Theorem 9.2.5.

Proof of Theorem 9.2.5. We compute the PAIM for L recursively:

- If $L \in F_0$, we can construct a PAIM for L in F_0 using Lemma 9.2.6.
- If $L \in F_i$ and $i \geqslant 1$, then $L = h(L' \cap \Psi^{-1}(S))$ for some $L' \subseteq X^*$ in G_{i-1}, a semilinear $S \subseteq X^{\oplus}$, and a morphism $h\colon X^* \to Y^*$. We compute a PAIM for L' in G_{i-1} and then use Lemma 9.2.20 to construct a PAIM for L.
- If $L \in G_i$, then $L = L(G)$ for an F_i-grammar G. We construct PAIM for the right-hand-sides of G and then using Lemma 9.2.19, we construct a PAIM for L in G_i.

□

9.3 Further applications of Parikh annotations

In this section, we demonstrate the utility of Parikh annotations by presenting two further applications. These applications are new proofs of existing results. In Chapter 10, we will put Parikh applications to use in yet another context by showing that the hierarchy $F_0 \subseteq G_0 \subseteq F_1 \subseteq \cdots$ is strict at every level.

9.3.1 Context-freeness of bounded languages

Our first application is a simple proof of a result of **GinsburgSpanier1966 [GinsburgSpanier1966]**, namely a characterization of those sets $S \subseteq \mathbb{N}^k$ for which the language

$$\{a_1^{n_1} \cdots a_k^{n_k} \mid (n_1, \ldots, n_k) \in S\}$$

is context-free. The characterization involves stratified sets. A set $P \subseteq \mathbb{N}^k$ is called *stratified* if

1. each element of P has at most two nonzero coordinates and
2. there are no indices $0 \leqslant r < s < t < u \leqslant k$ such that $x_r y_s x_t y_u \neq 0$ for some $(x_1, \ldots, x_k), (y_1, \ldots, y_k) \in P$.

The result we are concerned with is the following. It recently attracted attention in **[IbarraSeki2013, LerouxPenelleSutre2013]**. Let $X = \{a_1, \ldots, a_k\}$ and let $\tau\colon a_1^* \cdots a_k^* \to \mathbb{N}^k$ be the restriction of Ψ to $a_1^* \cdots a_k^*$.

Theorem 9.3.1 (GinsburgSpanier1966 [GinsburgSpanier1966]). *For $S \subseteq \mathbb{N}^k$, the set $\tau^{-1}(S)$ is context-free if and only if S is a finite union of linear sets with stratified sets of periods.*

It should be noted that the question of whether context-freeness of $\tau^{-1}(S)$ is decidable was left open in **[GinsburgSpanier1966]**. To the author's knowledge (and according to **[IbarraSeki2013, LerouxPenelleSutre2013]**), it still is.

It is easy to see that $\tau^{-1}(S)$ is context-free when S is a finite union of linear sets with stratified sets of periods. The proof for the opposite direction in **[GinsburgSpanier1966, GinsburgSpanier1964]** is not difficult, but involves a technical induction on k. Albeit rather involved itself, the existence of Parikh annotations provides a simple and conceptual proof of this latter direction. Hence, although the combination of Section 9.2 and the following is by no means simpler than **GinsburgSpanier1966**'s proof, it still illustrates the utility of Parikh annotations.

Suppose $L = \tau^{-1}(S)$ is context-free and $(K, C, P, (P_c)_{c \in C}, \varphi)$ is a context-free Parikh annotation for L, which exists by Theorem 9.2.5. We claim that for each $c \in C$, $\varphi(P_c)$ is stratified. Since $S = \Psi(L) = \bigcup_{c \in C} \varphi(c) + \varphi(P_c)^{\oplus}$, this clearly suffices. Let ρ be the rational transduction that, for each $a \in X$, removes from the input word the first $\varphi(c)(a)$ occurrences. If condition 1 were violated with $x_r x_s x_t \neq 0$ for $0 \leqslant r < s < t \leqslant k$ and some $\varphi(p) = (x_1, \ldots, x_k)$, then

$$\pi_{\{a_r, a_s, a_t\}}(\rho(K \cap c(\{p\} \cup X)^*)) = \{a_r^{mx_r} a_s^{mx_s} a_t^{mx_t} \mid m \in \mathbb{N}\}.$$

However, the left hand side is context-free, while the right-hand side is clearly not. If condition 2 were violated with $\varphi(p) = (x_1, \ldots, x_k)$, $\varphi(q) = (y_1, \ldots, y_k)$,

$p, q \in P_c$, and $0 \leqslant r < s < t < u \leqslant k$ with $x_r y_s x_t y_u \neq 0$, then condition 1 implies $x_m = y_n = 0$ for $m \notin \{r, t\}$, $n \notin \{s, u\}$ and hence

$$\pi_X(\rho(K \cap c(\{p, q\} \cup X)^*)) = \{a_r^{mx_r} a_s^{ny_s} a_t^{mx_t} a_u^{ny_u} \mid m, n \in \mathbb{N}\}.$$

Again, the left hand side is context-free and the right-hand side is clearly not.

9.3.2 Regularity of unambiguous constrained automata

Our next application is a new proof of a result of **CadilhacFinkelMcKenzie2012b [CadilhacFinkelMcK** It concerns *constrained automata*, a definitional variant of *Parikh automata*. The latter were introduced by **KlaedtkeRuess2003 [KlaedtkeRuess2003]** in order to obtain decidability of a fragment of Monadic Second Order Logic on finite words with cardinality constraints.

For general constrained automata (and hence Parikh automata), it is undecidable whether their accepted language is regular **[CadilhacFinkelMcKenzie2012a]**. However, **CadilhacFinkelMcKenzie2012b [CadilhacFinkelMcKenzie2012b]** have shown that in the case of *unambiguous constrained automata*, this problem becomes decidable. While the proof of **CadilhacFinkelMcKenzie2012b** is structurally similar, the one presented here is arguably conceptually simpler: Although it requires the somewhat involved machinery of Parikh annotations, the result is a relatively straightforward consequence of their existence for regular languages (with the slight extension of pseudo-boundedness), which is not hard to see (Lemma 9.3.5). Note that the proof here does not rely on the complex construction of Parikh annotations for F, but is self-contained.

Constrained automata A *constrained automaton* is a pair (A, C), where the first component $A = (Q, X, E, q_0, F)$ is a finite automaton and $C \subseteq E^{\oplus}$ is a semilinear set (here, we regard E as an alphabet). A *run* of (A, C) is a sequence

$$r = (p_1, w_1, q_1)(p_2, w_2, q_2) \cdots (p_n, w_n, q_n) \in E^*$$

with $p_i = q_{i-1}$ for $1 < i \leqslant n$, $p_1 = q_0$, and $q_n \in F$. By $\omega(r)$, we denote the word $w_1 \cdots w_n$. The language *accepted by* (A, C) is

$$L(A, C) = \{\omega(r) \mid r \in \Psi^{-1}(C) \text{ is a run of } (A, C)\}.$$

(A, C) is said to be *unambiguous* if for each $w \in L(A, C)$, there is at most one run r with $\omega(r) = w$.

Observe that in their general form, constrained automata accept precisely the languages in $\mathsf{SLI}(\mathsf{Reg}) = \bigcup_{n \in \mathbb{N}} \mathsf{VA}(\mathbb{Z}^n)$, in other words, those of blind multicounter automata. Note that unambiguous constrained automata subsume deterministic blind multicounter automata. In fact, they are slightly more expressive **[CadilhacFinkelMcKenzie2012b]**.

As mentioned above, we prove the following result.

Theorem 9.3.2 (CadilhacFinkelMcKenzie2012b [CadilhacFinkelMcKenzie2012b]). *Given an unambiguous constrained automaton (A, C), it is decidable whether $L(A, C)$ is regular.*

We will use the following result by Ginsburg and Spanier.

Theorem 9.3.3 (GinsburgSpanier1966 [GinsburgSpanier1966, GinsburgSpanier1966a]). *Given a semilinear set $S \subseteq X^{\oplus}$, it is decidable whether $\Psi^{-1}(S)$ is regular. Moreover, if $X = \{a_1, \ldots, a_n\}$, then $\Psi^{-1}(S)$ is regular if and only if $\Psi^{-1}(S) \cap a_1^* \cdots a_n^*$ is regular.*

As in **[CadilhacFinkelMcKenzie2012b]**, the first step is to reduce the problem to the regularity problem for languages of the form $L \cap \Psi^{-1}(S)$, where $L \subseteq X^*$ is regular and $S \subseteq X^{\oplus}$ is semilinear.

Lemma 9.3.4. *Given an unambiguous constrained automaton (A, C), one can construct a regular language $R \subseteq Y^*$ and a semilinear set $S \subseteq Y^{\oplus}$ such that $L(A, C)$ is regular if and only if $R \cap \Psi^{-1}(S)$ is regular.*

Proof. Let (A, C) be an unambiguous constrained automaton with the automaton $A = (Q, X, E, q_0, F)$ and the semilinear set $C \subseteq E^{\oplus}$. Let $R \subseteq E^*$ be the set of runs of (A, C), which is regular. We claim that then $L(A, C)$ is regular if and only if $R \cap \Psi^{-1}(C)$ is regular.

The "if" direction is clear: $L(A, C) = \omega(R \cap \Psi^{-1}(C))$ and morphisms preserve regularity.

For the "only if" direction, let $T \subseteq X^* \times E^*$ be the rational transduction that outputs for each input $w \in X^*$, every run $r \in E^*$ with $\omega(r) = w$. Then clearly, $R \cap \Psi^{-1}(C) \subseteq T(L(A, C))$. Moreover, since (A, C) is unambiguous, we have $T(L(A, C)) \subseteq R \cap \Psi^{-1}(C)$: If $w \in L(A, C)$, then there is at most one run $r \in E^*$ with $\omega(r) = w$. This means, r must the run satisfying $r \in \Psi^{-1}(C)$. Thus, $R \cap \Psi^{-1}(C) = T(L(A, C))$ and regularity of $L(A, C)$ implies regularity of $R \cap \Psi^{-1}(C)$. □

The remaining task is to decide regularity of sets of the form $L \cap \Psi^{-1}(S)$. To this end, we construct a regular Parikh annotation $(K, C, P, (P_c)_{c \in C}, \varphi)$ for L and then, using Theorem 9.3.3, we decide whether the sets $\Psi^{-1}(S_c)$ are regular, where $S_c = \{\mu \in P_c^{\oplus} \mid \varphi(c + \mu) \in S\}$. It is not hard to see that regularity of $\Psi^{-1}(S_c)$ implies regularity of $L \cap \Psi^{-1}(S)$ for any Parikh annotation. In order to have the converse, we use Parikh annotations with the additional guarantee of pseudo-boundedness.

Pseudo-boundedness A Parikh annotation $(K, C, P, (P_c)_{c \in C}, \varphi)$ for L is said to be *pseudo-bounded* if on each of the sets P_c, one can establish a linear order $(P_c, <)$ such that

$$c p_1^* \cdots p_n^* \subseteq \pi_{C \cup P}(K),$$

where $P_c = \{p_1, \ldots, p_n\}$ and $p_1 < \cdots < p_n$.

In other words, in a pseudo-bounded PA, when projecting to the annotation alphabet $\{c\} \cup P_c$, every description of a Parikh image appears as some word from the bounded language $c p_1^* \cdots p_n^*$. It is not hard to see that regular languages admit pseudo-bounded Parikh annotations.

Lemma 9.3.5. *Given a regular language R, one can construct a regular pseudo-bounded Parikh annotation for R.*

Proof. Let $R \subseteq X^*$ be regular. The proof proceeds by structural induction with respect to a rational expression for R. The statement is trivial for singletons. Since pseudo-bounded PAs are easy to construct for union and concatenation, we only consider the case that $R = S^*$ and we have constructed a pseudo-bounded PA

$(K, C, P, (P_c)_{c\in C}, \varphi)$ for $S \subseteq X^*$. Then for each $c \in C$, $(P_c, <)$ is linearly ordered such that the pseudo-boundedness condition holds. Here, we may assume that the sets P_c are pairwise disjoint.

For each $D \subseteq C$, let c_D be a new symbol. Moreover, let $C' = \{c_D \mid D \subseteq C\}$ and $P'_{c_D} = D \cup \bigcup_{c\in D} P_c$, $P' = C \cup P$. Furthermore, for words $w \in (C \cup X \cup P)^*$ with $|w|_c \geqslant 1$ for each $c \in D$, let $\delta_D(w)$ be the word obtained from w by deleting the first occurrence of each $c \in D$. With this, let T_D be the rational transduction with

$$T_D(M) = \{c_D \delta_D(w) \mid w \in M,\ \pi_C(w) \in D^*,\ |w|_c \geqslant 1 \text{ for each } c \in D\}.$$

Moreover, let $\varphi'(c_D) = \sum_{c\in D} \varphi(c)$ and $\varphi'(x) = \varphi(x)$ for $x \in C \cup P$. We claim that $(K', C', P', (P'_{c_D})_{c_D\in C'}, \varphi')$ with $K' = \bigcup_{D\subseteq C} T_D(K^*)$ is a pseudo-bounded PA for S^*. We construct the linear order on P'_{c_D} as follows. Choose an arbitrary linear order $(D, <)$ on D. Moreover, let

$$(P'_{c_D}, <) = \sum_{c\in D} (P_c, <) + (D, <)$$

be the order theoretic sum of the $(P_c, <)$ for $c \in D$ and of $(D, <)$.

We claim that $(K', C', P', (P'_{c_D})_{c_D\in C'}, \varphi')$ is a pseudo-bounded PA for S^*. Since completeness is subsumed by pseudo-boundedness, it is not necessary to prove the former.

- *Projection property.* Clearly, $\pi_X(K') \subseteq \pi_X(K^*) = S^*$. In order to prove the other inclusion, suppose $w_1, \ldots, w_n \in S$. For each i, we find a $c_i u_i \in K$ with $\pi_X(c_i u_i) = w_i$. We define the set $D = \{c_1, \ldots, c_n\}$ and the word $w' = c_D \delta_D(c_1 u_1 \cdots c_n u_n)$. Then, we clearly have $w' \in K'$ and also $\pi_X(w') = w_1 \cdots w_n$. Thus, $\pi_X(K') = S^*$.

- *Counting property.* Let $c_D \delta_D(w) \in K'$. Then $w \in K^*$ and $|w|_c \geqslant 1$ for $c \in D$. The counting property of K yields $\varphi(\pi_{C\cup P}(w)) = \Psi(\pi_X(w))$. By definition of δ_D, we have

$$\begin{aligned}\varphi'(\pi_{C'\cup P'}(c_D \delta_D(w))) &= \varphi'(c_D) + \varphi'(\pi_{C\cup P}(w)) - \sum_{c\in D} \varphi'(c)\\ &= \varphi(\pi_{C\cup P}(w)) = \Psi(\pi_X(w))\\ &= \pi_X(c_D \delta_D(w)).\end{aligned}$$

- *Pseudo-boundedness.* Let $c_D \in C'$ and suppose $P'_{c_D} = \{q_1, \ldots, q_m\}$ is ordered so that $q_1 < \cdots < q_m$. Furthermore, let $\mu \in P'^{\oplus}_{c_D}$. Then we can write $\mu = (\sum_{c\in D} \mu_c) + \kappa$ for $\mu_c \in P_c^{\oplus}$ and $\kappa \in D^{\oplus}$. The pseudo-boundedness of K implies that for each $c \in D$, we can find some w_c in $(X \cup P)^*$ with $c w_c \in K$ and $\pi_{C\cup P}(w_c) = p_1^{\mu_c(p_1)} \cdots p_n^{\mu_c(p_n)}$, where $P_c = \{p_1, \ldots, p_n\}$, $p_1 < \cdots < p_n$. Furthermore, for each $c \in D$, there is a $v_c \in K$ with $\pi_{C\cup P}(v_c) = c$. Now if $D = \{c_1, \ldots, c_k\}$, $c_1 < \cdots < c_k$, then

$$c_1 w_{c_1} \cdots c_k w_{c_k} v_{c_1}^{\kappa(c_1)} \cdots v_{c_k}^{\kappa(c_k)} \in K^*$$

  and hence

$$w = c_D w_{c_1} \cdots w_{c_k} v_{c_1}^{\kappa(c_1)} \cdots v_{c_k}^{\kappa(c_k)} \in K'.$$

Then clearly $\pi_{C' \cup P'}(w) = c_D q_1^{\mu(q_1)} \cdots q_m^{\mu(q_m)}$. This proves the inclusion $c_D q_1^* \cdots q_m^* \subseteq \pi_{C' \cup P'}(K')$.

□

The pseudo-bounded Parikh annotations now allow us to reduce the regularity problem for languages $R \cap \Psi^{-1}(S)$ to languages of the form $\Psi^{-1}(S)$.

Lemma 9.3.6. *Given a regular language $R \subseteq X^*$ and a semilinear set $S \subseteq X^\oplus$, one can construct semilinear sets $S_1, \ldots, S_n$, $S_i \subseteq Y_i^\oplus$, such that $R \cap \Psi^{-1}(S)$ is regular if and only if each $\Psi^{-1}(S_i)$ is regular for $1 \leqslant i \leqslant n$.*

Proof. Let $R \subseteq X^*$ be regular, $S \subseteq X^\oplus$ be semilinear, and $(K, C, P, (P_c)_{c \in C}, \varphi)$ be a pseudo-bounded Parikh annotation for R. Furthermore, note that the set $S_c = \{\mu \in P_c^\oplus \mid \varphi(c + \mu) \in S\}$ is Presburger definable and hence effectively semilinear. We shall prove that the sets $S_c \subseteq P_c^\oplus$ have the desired property.

Suppose $R \cap \Psi^{-1}(S)$ is regular and $c \in C$. Then the language

$$K_c = \{cv \in K \mid \pi_X(cv) \in R \cap \Psi^{-1}(S)\}$$

inherits regularity from K and $R \cap \Psi^{-1}(S)$. Since $\pi_X(K) = R$, we have

$$\begin{aligned} K_c &= \{v \in K \mid \pi_X(cv) \in \Psi^{-1}(S)\} \\ &= \{v \in K \mid \varphi(\pi_{C \cup P}(cv)) \in \Psi^{-1}(S)\} \\ &= \{v \in K \mid \pi_P(v) \in \Psi^{-1}(S_c)\} \end{aligned}$$

Let $P_c = \{p_1, \ldots, p_n\}$ with $p_1 < \cdots < p_n$. The pseudo-boundedness of K implies

$$\Psi(\pi_P(K_c) \cap p_1^* \cdots p_n^*) = \Psi(\pi_P(K_c)) = S_c.$$

This means the set $B_c = \pi_P(K_c) \cap p_1^* \cdots p_n^* \subseteq p_1^* \cdots p_n^*$ is regular and satisfies $\Psi(B_c) = S_c$. According to Theorem 9.3.3, this implies regularity of $\Psi^{-1}(S_c)$. Now suppose $\Psi^{-1}(S_c)$ is regular for each $c \in C$. Then regularity of $R \cap \Psi^{-1}(S)$ follows because

$$\begin{aligned} R \cap \Psi^{-1}(S) &= \{\pi_X(w) \mid w \in K,\ \Psi(\pi_X(w)) \in S\} \\ &= \{\pi_X(cv) \mid c \in C,\ cv \in K,\ \Psi(\pi_P(v)) \in S_c\} \\ &= \{\pi_X(cv) \mid c \in C,\ cv \in K,\ \pi_P(v) \in \Psi^{-1}(S_c)\}. \end{aligned}$$

□

We are now ready to prove Theorem 9.3.2.

Proof of Theorem 9.3.2. The theorem follows directly from Lemmas 9.3.4 and 9.3.6 and Theorem 9.3.3. □

9.4 Conclusion

The computation of downward closures is a task that is still little understood, but promises applications and is theoretically appealing. In this chapter, we have added our class F to the list of language classes for which downward closures

are computable. On the one hand, this class appears to be relatively large and exhibits pleasant closure properties—it is a Presburger closed full AFL. On the other hand, it permits modeling recursive programs with numeric data types (see Chapter 12 for details).

The technique for proving this main result is the concept of Parikh annotations. They can be thought of as giving finite state transducers access to Parikh decompositions and hence endow the individual levels of our hierarchy with more closure properties.

We have also applied Parikh annotations to provide new proofs of existing results. On the one hand, this concerns a characterization of strictly bounded context-free languages by **GinsburgSpanier1966** [**GinsburgSpanier1966**]. On the other hand, we presented a new proof for the decidability of the regularity problem for unambiguous constrained automata, which had originally been shown by **CadilhacFinkelMcKenzie2012b** [**CadilhacFinkelMcKenzie2012b**]. These applications demonstrate the utility of Parikh annotations.

The results of this section have appeared in [**Zetzsche2015a**].

Related work As mentioned above, methods for computing downward closures have been obtained for other language classes. For context-free languages and, slightly more general, for algebraic extensions, this has been shown by **vanLeeuwen1978** [**vanLeeuwen1978**]. A different approach for the context-free languages was then presented by **Courcelle1991** [**Courcelle1991**]. For 0L-systems and context-free FIFO rewriting systems, computability has been established by **AbdullaBoassonBouajjani2001** [**AbdullaBoassonBouajjani2001**]. Finally, **HabermehlMeyerWimmel** were able to show computability for the Petri net languages.

Furthermore, it was shown by **GruberHolzerKutrib2007** [**GruberHolzerKutrib2007**] that downward closures are not computable for Church-Rosser languages and **Mayr2003** [**Mayr2003**] has shown that reachability sets of lossy channel systems cannot be computed.

Complexity issues, both descriptional and computational, have been considered by **GruberHolzerKutrib2009** [**GruberHolzerKutrib2009**], **Okhotin2010** [**Okhotin2010**], **KarandikarNiewerthSchnoebelen2016** [**KarandikarNiewerthSchnoebelen2016**], and **BachmeierLuttenbergerSchlund2015** [**BachmeierLuttenbergerSchlund2015**].

Open problems

1. Of course, it remains the task of providing more language classes with methods to compute downward closures. In particular, it is not clear for which graph monoids downward closures are computable. The result of **HabermehlMeyerWimmel2010** [**HabermehlMeyerWimmel2010**], together with **vanLeeuwen1978**'s [**vanLeeuwen1978**], implies computability for $\mathbb{B} * \mathbb{B}^n$ and the result here covers the monoids in SL. Hence, the question is: For which graphs Γ is there an algorithm to compute downward closures of $\mathsf{VA}(\mathbb{M}\Gamma)$? It should be mentioned that the author of this work is currently developing a new approach to downward closure computation that will likely be applicable to a wide range of language classes. While this new approach subsumes the computability for F, its complexity is likely much higher than the algorithm presented here.

2. Furthermore, the decidability status of the context-freeness of sets of the form $\{a_1^{n_1} \cdots a_k^{n_k} \mid (n_1, \ldots, n_k) \in S\}$ for a given semilinear $S \subseteq \mathbb{N}^k$ remains open. Equivalent decision problems have been presented by **IbarraSeki2013** [**IbarraSeki2013**]. For partial solutions, see [**Lisovik1996**].

Acknowledgements I would like to thank Roland Meyer for discussions on the computability and applications of downward closures. I am also grateful to the anonymous referees for STACS 2015, who made valuable suggestions regarding the presentation. Finally, I am thankful to Sylvain Schmitz and Philippe Schnoebelen, who pointed out to me that simple regular expressions had been discovered before [**Abdulla2004**] (and were called ideal decompositions in the earlier work [**Jullien1969**]).

Chapter 10

Non-expressibility results

10.1 Introduction

A crucial part of studying the expressiveness of automata models is to show that certain languages cannot be accepted by a particular model. Moreover, the proofs for such results often provide an intuition for the limits of the model at hand. While in Chapters 3, 6, and 7, we have investigated when the classes $\mathsf{VA}(M)$ are confined to regular, context-free, or semilinear languages, this is rarely sufficient when we want to compare the expressiveness of two monoids. Therefore, in this chapter, we turn to results that prove certain languages not to be accepted.

First, we survey a few known results. Then, we prove here that all inclusions $\mathsf{F}_0 \subseteq \mathsf{G}_0 \subseteq \mathsf{F}_1 \subseteq \mathsf{G}_1 \subseteq \cdots$ in our hierarchy, which exhausts graph-defined storage mechanisms that guarantee semilinearity, are strict. The following results all pertain to language classes induced by graph monoids.

The main contribution in this chapter, the strictness of the hierarchy, appeared in [**Zetzsche2015a**].

Known results The following was shown by Latteux [**Latteux1977**]. Alternative proofs have been obtained by Fernau and Stiebe [**FernauStiebe2002a**] and by **ClearyElderOstheimer2006** [**ClearyElderOstheimer2006**].

Theorem 10.1.1 (Latteux [**Latteux1977**])**.** *For each* $k \in \mathbb{N}$, $\mathsf{VA}(\mathbb{Z}^k) \subsetneq \mathsf{VA}(\mathbb{Z}^{k+1})$.

It can also be derived from the material in this work. Since $\mathsf{VA}(\mathbb{Z}^k)$ equals $\mathsf{VA}^+(\mathbb{Z}^k)$ (Theorem 8.1.1), it suffices to show $\mathsf{VA}^+(\mathbb{Z}^k) \subsetneq \mathsf{VA}^+(\mathbb{Z}^{k+1})$. Moreover, the proof of Lemma 8.5.2 yields that for $L \in \mathsf{VA}^+(\mathbb{Z}^k)$, the function f_L is bounded from above by a polynomial of degree k. If we then choose

$$L_k = \{a_1^{n_1} \cdots a_k^{n_k} b_1^{n_1} \cdots b_k^{n_k} \mid n_1, \ldots, n_k \in \mathbb{N}\},$$

it is easy to see that $L_k \in \mathsf{VA}^+(\mathbb{Z}^k)$ and that f_{L_k} is also bounded from below by a polynomial of degree k. Hence, $L_{k+1} \in \mathsf{VA}^+(\mathbb{Z}^{k+1}) \setminus \mathsf{VA}^+(\mathbb{Z}^k)$. Coming back to our motivation for studying non-expressibility results, namely getting an intuition for expressiveness, this teaches us that the number of counters determines the magnitude of growth of the number of configurations we must be able to distinguish in order to accept a language.

Using the same argument, one can show that $L_{k+1} \in \mathsf{VA}^+(\mathbb{B}^{k+1}) \setminus \mathsf{VA}^+(\mathbb{B}^k)$. However, since we have no ε-removal for $\mathbb{B}^k$, this does not tell us whether the inclusions $\mathsf{VA}(\mathbb{B}^k) \subseteq \mathsf{VA}(\mathbb{B}^{k+1})$ are all strict and to the best of the author's knowledge, this is still an open problem **[Jantzen1979]**. However, according to Lemma 2.3.8, if these were not all strict, then there would be a fixed $n \in \mathbb{N}$ such that $\mathsf{VA}(\mathbb{B}^n)$ contains $\mathsf{VA}(\mathbb{B}^k)$ for all $k \in \mathbb{N}$. This seems very unlikely.

A non-expressibility result for valence grammars has been obtained by **Vicolov1994** and re-proved by **FernauStiebe2002a [FernauStiebe2002a]** using an iteration lemma based on linear algebraic arguments.

Theorem 10.1.2 (Vicolov1994 [Vicolov1994]). *For each* $k \in \mathbb{N}$, $\mathsf{VG}(\mathbb{Z}^k) \subsetneq \mathsf{VG}(\mathbb{Z}^{k+1})$.

A simple construction shows that for each $k \in \mathbb{N}$, the class $\mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{Z}^k)$ equals $\mathsf{VG}(\mathbb{Z}^k)$ (see, for example, **[Hoogeboom2002]**). We can therefore conclude the following from Theorem 10.1.2.

Corollary 10.1.3. *For each* $k \in \mathbb{N}$, $\mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{Z}^k) \subsetneq \mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{Z}^{k+1})$.

Strictness of the hierarchy Using Parikh annotations with insertion markers, one can show that the inclusions $F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \cdots$ in the hierarchy are, in fact, all strict. More precisely, we have the following.

Theorem 10.1.4. *Let* X_i *and* Y_i *for* $i \in \mathbb{N}$ *be the alphabets* $X_0 = \emptyset$, $Y_i = X_i \cup \{\#_i\}$, $X_{i+1} = Y_i \cup \{a_{i+1}, b_{i+1}, c_{i+1}\}$. *Moreover, define the sets* $U_i \subseteq X_i^*$ *and* $V_i \subseteq Y_i^*$ *as* $U_0 = \{\varepsilon\}$ *and*

$$V_i = (U_i \#_i)^*, \qquad U_{i+1} = V_i \sqcup \{a_{i+1}^n b_{i+1}^n c_{i+1}^n \mid n \geqslant 0\}$$

for $i \geqslant 0$. *Then* $V_i \in G_i \setminus F_i$ *and* $U_{i+1} \in F_{i+1} \setminus G_i$.

Recall that the levels correspond to the alternation of the two transformations *building stacks* and *adding blind counters* (see the proof of Proposition 7.1.2) of storage mechanisms. Therefore, Theorem 10.1.4 means in particular that statements about stacked counter automata, such as the ε-removal in Chapter 8, must indeed consider arbitrarily deep nestings of these two transformations. Of course, this is also true for algorithms that work directly with the hierarchy, such as the computation of downward closures in Chapter 9.

Before we prove Theorem 10.1.4 in the next section, let us record some consequences. Exhibiting a strict hierarchy of full trios that exhausts a language class always implies that the language class cannot be a principal full trio. Intuitively, and as explained before Corollary 2.3.5, this means that there is no fixed set of operations that suffices to accept all languages.

Corollary 10.1.5. F *is not a principal full trio. In particular, there is no finitely generated monoid* M *with* $\mathsf{VA}(M) = F$.

Proof. Suppose F were generated as a full trio by a language $L \in F$. Then $L \in F$ means that $L \in F_i$ for some $i \in \mathbb{N}$. Since F_i is a full trio, this implies $F \subseteq F_i$, contradicting Theorem 10.1.4. The second statement follows from Corollary 2.3.5. □

By paying attention to what storage we actually need to accept the separating languages U_i and V_i, we can conclude that strictness is already achieved by

adding two counters (as opposed to some finite number) in each transformation. In the following corollary, we use the notation

$$\mathsf{Z}(\mathsf{M}) = \mathsf{M} \times \mathbb{Z}, \qquad \mathsf{B}(\mathsf{M}) = \mathbb{B} * \mathsf{M}.$$

For example, we have

$$\mathsf{Z}^i\mathsf{B}(\mathsf{M}) = (\mathbb{B} * \mathsf{M}) \times \mathbb{Z}^i, \qquad (\mathsf{ZB})^2(\mathsf{M}) = (\mathbb{B} * ((\mathbb{B} * \mathsf{M}) \times \mathbb{Z})) \times \mathbb{Z}.$$

Corollary 10.1.6. *Let* $i \in \mathbb{N}$ *and* $n_1, \dots, n_i \in \mathbb{N}$. *Then we have*

$$\mathsf{VA}(\mathsf{B}(\mathsf{Z}^2\mathsf{B})^i(\mathbf{1})) \setminus \mathsf{VA}(\mathsf{Z}^{n_1}\mathsf{B} \cdots \mathsf{Z}^{n_i}\mathsf{B}(\mathbf{1})) \neq \emptyset, \tag{10.1}$$

$$\mathsf{VA}(\mathsf{Z}^2\mathsf{B}(\mathsf{Z}^2\mathsf{B})^i(\mathbf{1})) \setminus \mathsf{VA}(\mathsf{B}\mathsf{Z}^{n_1}\mathsf{B} \cdots \mathsf{Z}^{n_i}\mathsf{B}(\mathbf{1})) \neq \emptyset. \tag{10.2}$$

Proof. Observe that $\mathsf{U}_0 \in \mathsf{VA}(\mathbf{1})$ and if $\mathsf{U}_i \in \mathsf{VA}(\mathsf{M})$, then $\mathsf{V}_i \in \mathsf{VA}(\mathsf{B}(\mathsf{M}))$. Moreover, if $\mathsf{V}_i \in \mathsf{VA}(\mathsf{M})$, then $\mathsf{U}_{i+1} \in \mathsf{VA}(\mathsf{Z}^2(\mathsf{M}))$. Therefore,

$$\mathsf{V}_i \in \mathsf{VA}(\mathsf{B}(\mathsf{Z}^2\mathsf{B})^i(\mathbf{1})), \qquad \mathsf{U}_{i+1} \in \mathsf{VA}(\mathsf{Z}^2\mathsf{B}(\mathsf{Z}^2\mathsf{B})^i(\mathbf{1})) \tag{10.3}$$

for all $i \in \mathbb{N}$.

Furthermore, if $\mathsf{VA}(\mathsf{M}) \subseteq \mathsf{F}_i$ for some $i \in \mathbb{N}$, then $\mathsf{VA}(\mathsf{B}(\mathsf{M})) \subseteq \mathsf{Alg}(\mathsf{F}_i) = \mathsf{G}_i$ (see Theorem 2.6.6). If $\mathsf{VA}(\mathsf{M}) \subseteq \mathsf{G}_i$, then $\mathsf{VA}(\mathsf{Z}^n(\mathsf{M})) \subseteq \mathsf{SLI}(\mathsf{G}_i) = \mathsf{F}_{i+1}$ (see Proposition 2.5.3). We have $\mathsf{VA}(\mathsf{B}(\mathbf{1})) = \mathsf{VA}(\mathbb{B}) \subseteq \mathsf{G}_0$ and hence

$$\mathsf{VA}(\mathsf{B}\mathsf{Z}^{n_1}\mathsf{B} \cdots \mathsf{Z}^{n_i}\mathsf{B}(\mathbf{1})) \subseteq \mathsf{G}_i \qquad \text{for } i \geqslant 0,$$
$$\mathsf{VA}(\mathsf{Z}^{n_1}\mathsf{B} \cdots \mathsf{Z}^{n_i}\mathsf{B}(\mathbf{1})) \subseteq \mathsf{F}_i \qquad \text{for } i \geqslant 1.$$

Since $\mathsf{U}_{i+1} \notin \mathsf{G}_i$ and $\mathsf{V}_i \notin \mathsf{F}_i$, together with (10.3), this proves (10.2) for $i \geqslant 0$ and (10.1) for $i \geqslant 1$.

Note that (10.1) also holds for $i = 0$: This amounts to showing that $\mathsf{VA}(\mathbf{1})$ is strictly included in $\mathsf{VA}(\mathbb{B})$, which is trivial since $\mathsf{VA}(\mathbb{B})$ contains the non-regular language $\{a^n b^n \mid n \geqslant 0\}$. □

Furthermore, we can conclude that in a stacked counter mechanism whose outermost applied transformation was *building stacks*, already adding a single blind counter yields strictness.

Corollary 10.1.7. *For every* $\mathsf{M} \in \mathsf{SC}^-$, *we have* $\mathsf{VA}(\mathbb{B} * \mathsf{M}) \subsetneq \mathsf{VA}((\mathbb{B} * \mathsf{M}) \times \mathbb{Z})$.

Proof. Suppose there were an $\mathsf{M} \in \mathsf{SC}^-$ with $\mathsf{VA}((\mathbb{B} * \mathsf{M}) \times \mathbb{Z}) = \mathsf{VA}(\mathbb{B} * \mathsf{M})$. Since $\mathsf{M} \in \mathsf{SC}^-$, there is an $i \in \mathbb{N}$ with $\mathsf{VA}(\mathbb{B} * \mathsf{M}) \subseteq \mathsf{F}_i$ (see the proof of Proposition 7.1.2). If $\mathsf{M} = \mathbf{1}$, our assumption reads $\mathsf{VA}(\mathbb{B} \times \mathbb{Z}) = \mathsf{VA}(\mathbb{B})$, which contradicts the fact that the class $\mathsf{VA}(\mathbb{B} \times \mathbb{Z})$ contains the non-context-free language $\{a^n b^n c^n \mid n \geqslant 0\}$, while $\mathsf{VA}(\mathbb{B})$ contains only context-free languages. Hence, we may assume $\mathsf{M} \neq \mathbf{1}$.

According to Theorem 2.6.6, this implies $\mathsf{Alg}(\mathsf{VA}(\mathbb{B} * \mathsf{M})) = \mathsf{VA}(\mathbb{B} * \mathsf{M})$. By Lemma 2.3.8, we have $\mathsf{VA}((\mathbb{B} * \mathsf{M}) \times \mathbb{Z}^n) = \mathsf{VA}(\mathbb{B} * \mathsf{M})$ for every $n \in \mathbb{N}$ and hence $\mathsf{VA}(\mathbb{B} * \mathsf{M}) = \bigcup_{n \geqslant 0} \mathsf{VA}((\mathbb{B} * \mathsf{M}) \times \mathbb{Z}^n) = \mathsf{SLI}(\mathsf{VA}(\mathbb{B} * \mathsf{M}))$. In other words, $\mathsf{VA}(\mathbb{B} * \mathsf{M})$ is closed under taking the algebraic extension and under imposing Presburger constraints. Since $\mathsf{F}_0 \subseteq \mathsf{VA}(\mathbb{B} * \mathsf{M})$, this implies $\mathsf{F} \subseteq \mathsf{VA}(\mathbb{B} * \mathsf{M}) \subseteq \mathsf{F}_i$, contradicting Theorem 10.1.4. □

As an amusing byproduct of the proof of Theorem 10.1.4, we will obtain the following result. While it is well-known that the classes $\mathsf{VA}(\mathbb{Z}^n)$ are not closed under marked Kleene iteration, this states that the marked Kleene iteration is available only in trivial cases. Note that this is not true for the ordinary Kleene iteration, since the language $L = \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$ belongs to $\mathsf{VA}(\mathbb{Z})$ and satisfies $L = L^*$.

Corollary 10.1.8. *Let $L \subseteq X^*$ and $\# \notin X$. If $(L\#)^* \in \mathsf{VA}(\mathbb{Z}^n)$, then L is regular.*

Note that Lemma 11.3.3 states that the same is true of torsion groups (as opposed to $\mathbb{Z}^n$). However, Lemma 11.3.3 and Corollary 10.1.8 use quite different arguments.

10.2 Strictness of the hierarchy

This section is devoted to the proof of Theorem 10.1.4. It is of course easy to see that $\mathsf{F}_0 \subsetneq \mathsf{G}_0 \subsetneq \mathsf{F}_1$, since F_0 contains only finite sets, G_0 is the class of context-free languages, and F_1 contains, for example, $\{a^n b^n c^n \mid n \geqslant 0\}$. In order to prove strictness at higher levels, we present two transformations: The first turns a language from $\mathsf{F}_i \setminus \mathsf{G}_{i-1}$ into one in $\mathsf{G}_i \setminus \mathsf{F}_i$ (Proposition 10.2.1) and the second turns one from $\mathsf{G}_i \setminus \mathsf{F}_i$ into one in $\mathsf{F}_{i+1} \setminus \mathsf{G}_i$ (Proposition 10.2.3).

The essential idea of Proposition 10.2.1 is the following. For the sake of simplicity, assume $(L\#)^* = L' \cap \Psi^{-1}(S)$ for $L' \in \mathcal{C}$, $L' \subseteq (X \cup \{\#\})^*$. Consider a PAIM $(K', C, P, (P_c)_{c\in C}, \varphi, \diamond)$ for L' in $\mathcal{C}$. Using a rational transduction, we obtain from K' a language $\hat{L} \subseteq (X \cup \{\#, \diamond\})^*$ in $\mathcal{C}$ such that every member of $\hat{L}$ admits an insertion at $\diamond$ that yields a word from $(L\#)^* = L' \cap \Psi^{-1}(S)$. Using rational transductions again, we can then pick all words that appear between two # in some member of $\hat{L}$ and contain no $\diamond$. Since there is a bound on the number of $\diamond$ in K' (and hence in $\hat{L}$), every word from L has to occur in this way. On the other hand, since inserting at $\diamond$ yields a word in $(L\#)^*$, every such word without $\diamond$ must be in L.

Proposition 10.2.1. *Let $\mathcal{C}$ be a full trio such that every language in $\mathcal{C}$ has a PAIM in $\mathcal{C}$. Moreover, let X be an alphabet with $\# \notin X$. If $(L\#)^* \in \mathsf{SLI}(\mathcal{C})$ for $L \subseteq X^*$, then $L \in \mathcal{C}$.*

Proof. Let $Y = X \cup \{\#\}$. Suppose $(L\#)^* \in \mathsf{SLI}(\mathcal{C})$. This means that we can write $(L\#)^* = h(L' \cap \Psi^{-1}(S))$ for some $L' \subseteq Z^*$, a semilinear $S \subseteq Z^\oplus$, and a morphism $h\colon Z^* \to Y^*$. Since $\mathcal{C}$ has PAIMs, there is a PAIM $(K, C, P, (P_c)_{c\in C}, \varphi, \diamond)$ for L' in $\mathcal{C}$. Let $S_c = \{\mu \in P_c^\oplus \mid \varphi(c+\mu) \in S\}$. Moreover, let g be the morphism with

$$\begin{aligned} g\colon (C \cup Z \cup P \cup \{\diamond\})^* &\longrightarrow (Y \cup \{\diamond\})^* \\ z &\longmapsto h(z) && \text{for } z \in Z, \\ x &\longmapsto \varepsilon && \text{for } x \in C \cup P, \\ \diamond &\longmapsto \diamond. \end{aligned}$$

Finally, we need the rational transduction $T \subseteq (Y \cup \{\diamond\})^* \times X^*$ with

$$T(M) = \{s \in X^* \mid r\#s\#t \in M \text{ for some } r, t \in (Y \cup \{\diamond\})^*\}.$$

We claim that

$$L = T(\hat{L}), \qquad \text{where} \qquad \hat{L} = \{g(cw) \mid c \in C,\ cw \in K,\ \pi_P(w) \in \Psi^{-1}(S_c{\downarrow})\}.$$

According to Corollary 2.8.3, the language $\Psi^{-1}(S_c{\downarrow})$ is regular, meaning $\hat{L} \in \mathcal{C}$ and hence $T(\hat{L}) \in \mathcal{C}$. Thus, proving $L = T(\hat{L})$ establishes the proposition.

We begin with the inclusion $T(\hat{L}) \subseteq L$. Let $s \in T(\hat{L})$ and hence $r\#s\#t = g(cw)$ for $r, t \in (Y \cup \{\diamond\})^*$, $c \in C$, $cw \in K$ and $\pi_P(w) \in \Psi^{-1}(S_c{\downarrow})$. The latter means there is a $\mu \in P_c^{\oplus}$ such that $\Psi(\pi_P(w)) + \mu \in S_c$ and hence

$$\Psi(\pi_Z(cw)) + \varphi(\mu) = \varphi(c + \Psi(\pi_P(w)) + \mu) \in S.$$

By the insertion property of K, there is a $\nu \in L'$ with $\pi_{Z\cup\{\diamond\}}(cw) \preceq_\diamond \nu$ and $\Psi(\nu) = \Psi(\pi_Z(cw)) + \varphi(\mu)$. This means $\Psi(\nu) \in S$ and thus $\nu \in L' \cap \Psi^{-1}(S)$ and hence $g(\nu) = h(\nu) \in (L\#)^*$. Since $g(\diamond) = \diamond$, the relation $\pi_{Z\cup\{\diamond\}}(cw) \preceq_\diamond \nu$ implies

$$r\#s\#t = g(cw) = g(\pi_{Z\cup\{\diamond\}}(cw)) \preceq_\diamond g(\nu) \in (L\#)^*.$$

However, $\diamond$ does not occur in s, meaning $\#s\# \in \#X^*\#$ is a factor of $g(\nu) \in (L\#)^*$ and hence $s \in L$. This proves $T(\hat{L}) \subseteq L$.

In order to show $L \subseteq T(\hat{L})$, suppose $s \in L$. The boundedness property of K means there is a bound $k \in \mathbb{N}$ with $|w|_\diamond \leqslant k$ for every $w \in K$. Consider the word $\nu = (s\#)^{k+2}$. Since $\nu \in (L\#)^*$, we find a $\nu' \in L' \cap \Psi^{-1}(S)$ with $\nu = h(\nu')$. This, in turn, means there is a $cw \in K$ with $c \in C$ and $\pi_Z(cw) = \nu'$. Then

$$\varphi(c + \Psi(\pi_P(w))) = \varphi(\pi_{C\cup P}(cw)) = \Psi(\pi_Z(cw)) = \Psi(\nu') \in S$$

and hence $\Psi(\pi_P(w)) \in S_c \subseteq S_c{\downarrow}$. Therefore, $g(cw) \in \hat{L} \subseteq (Y \cup \{\diamond\})^*$. Note that g agrees with $h(\pi_Z(\cdot))$ on all symbols but $\diamond$, which is fixed by the former and erased by the latter. Since $h(\pi_Z(cw)) = h(\nu') = \nu = (s\#)^{k+2}$, the word $g(cw)$ is obtained from $(s\#)^{k+1}$ by inserting occurrences of $\diamond$. In fact, it is obtained by inserting at most k of them since $|g(cw)|_\diamond = |cw|_\diamond \leqslant k$. This means $g(cw)$ has at least one factor $\#s\# \in \#X^*\#$ and hence $s \in T(g(cw)) \subseteq T(\hat{L})$. This completes the proof of $L = T(\hat{L})$ and thus of the proposition. □

Proposition 10.2.1 now allows us to prove that $(L\#)^*$ can only be accepted by a blind multicounter automaton when L is regular.

Proof of Corollary 10.1.8. In Lemma 9.3.5, we have seen that regular languages admit Parikh annotations. It is not hard to extend the proof so as to construct Parikh annotations with insertion markers. Hence, applying Proposition 10.2.1 to $\mathcal{C} = \mathsf{Reg}$ and noting that $\bigcup_{n\in\mathbb{N}} \mathsf{VA}(\mathbb{Z}^n)$ equals $\mathsf{SLI}(\mathsf{Reg})$ (Proposition 2.5.3) yields the corollary. □

In order to prove Proposition 10.2.3, we need a new concept. A bursting grammar is one in which essentially the entire word is generated in a single application of a production. Here, 'essentially' means: aside from a subsequent replacement by terminal words of bounded length.

Bursting grammars Let $\mathcal{C}$ be a language class and $k \in \mathbb{N}$. A $\mathcal{C}$-grammar G is called k-*bursting* if for every derivation tree t for G and every node x of t we have: $|\mathsf{yield}(x)| > k$ implies $\mathsf{yield}(x) = \mathsf{yield}(t)$. A grammar is said to be *bursting* if it is k-bursting for some $k \in \mathbb{N}$.

In a bursting $\mathcal{C}$-grammar, we can, using a rational transduction, extract $L(G)$ (finitely many exceptions aside) from those right-hand sides that essentially generate the entire word.

Lemma 10.2.2. *If $\mathcal{C}$ is a union closed full semi-trio and G a bursting $\mathcal{C}$-grammar, then $L(G) \in \mathcal{C}$.*

Proof. Suppose $G = (N, T, P, S)$ is k-bursting. Let $\sigma\colon (N \cup T)^* \to \mathcal{P}(T^*)$ be the substitution with $\sigma(x) = \{w \in T^{\leqslant k} \mid x \Rightarrow_G^* w\}$ for $x \in N \cup T$. Since $\sigma(x)$ is finite for each $x \in N \cup T$, there is clearly a locally finite rational transduction τ with $\tau(M) = \sigma(M)$ for every language $M \subseteq (N \cup T)^*$. In particular, $\sigma(M) \in \mathcal{C}$ whenever $M \in \mathcal{C}$. Let $R \subseteq N$ be the set of reachable nonterminals. We claim that

$$L(G) \cap T^{>k} = \bigcup_{A \in R} \bigcup_{A \to L \in P} \sigma(L) \cap T^{>k}. \tag{10.4}$$

This clearly implies $L(G) \cap T^{>k} \in \mathcal{C}$. Furthermore, since $\mathcal{C}$ is a union closed full semi-trio and thus closed under adding finite sets of words, it even implies $L(G) \in \mathcal{C}$ and hence the lemma.

We start with the inclusion "$\subseteq$". Suppose $w \in L(G) \cap T^{>k}$ and let t be a derivation tree for G with $\mathrm{yield}(t) = w$. Since $|w| > k$, t clearly has at least one node x with $|\mathrm{yield}(x)| > k$. Let y be maximal among these nodes (i.e. such that no descendent of y has a yield of length $> k$). Since G is k-bursting, this means $\mathrm{yield}(y) = w$. Furthermore, each child c of y has $|\mathrm{yield}(c)| \leqslant k$. Thus, if A is the label of y, then A is reachable and there is a production $A \to L$ with $w \in \sigma(L)$. Hence, w is contained in the right-hand side of (10.4).

In order to show "$\supseteq$" of (10.4), suppose $w \in \sigma(L) \cap T^{>k}$ for some $A \to L \in P$ and a reachable $A \in N$. By the definition of σ, we have $A \Rightarrow_G^* w$. Since A is reachable, there is a derivation tree t for G with an A-labeled node x such that $\mathrm{yield}(x) = w$. Since G is k-bursting and $|w| > k$, this implies

$$w = \mathrm{yield}(x) = \mathrm{yield}(t) \in L(G)$$

and thus $w \in L(G) \cap T^{>k}$. □

We can now use Lemma 10.2.2 and Lemma 9.2.12 to prove the next proposition. The essential idea for Proposition 10.2.3 is the following. We construct a $\mathcal{C}$-grammar G' for L by removing from a $\mathcal{C}$-grammar G for

$$M = (L \sqcup \{a^n b^n c^n \mid n \geqslant 0\}) \cap a^*(bX)^*c^*$$

all terminals a, b, c. If $Y = X \cup \{a, b, c\}$, the morphisms $\alpha, \beta\colon Y^* \to \mathbb{Z}$ with $\alpha(w) = |w|_a - |w|_b$ and $\beta(w) = |w|_b - |w|_c$ always yield 0 on words in M. Therefore, Lemma 9.2.12 provides a bound on the values of α and β on yields of subtrees in derivation trees of G. This allows us to prove that G' must be bursting: If a node x generates a sufficiently long word from X^*, then the corresponding node in G must have a large number of b's below it. The boundedness of α and β then implies that it also has at least one a and one c below it, meaning that x already generates the whole word.

Proposition 10.2.3. *Let $\mathcal{C}$ be a union closed full semi-trio and let $a, b, c \notin X$ and $L \subseteq X^*$. If $L \sqcup \{a^n b^n c^n \mid n \geqslant 0\} \in \mathrm{Alg}(\mathcal{C})$, then $L \in \mathcal{C}$.*

Proof. Let $K = L \sqcup \{a^n b^n c^n \mid n \geqslant 0\}$. If $K \in \mathrm{Alg}(\mathcal{C})$, then also the intersection $M = K \cap a^*(bX)^*c^*$ belongs to $\mathrm{Alg}(\mathcal{C})$. Hence, let $M = L(G)$ for a reduced $\mathcal{C}$-grammar $G = (N, T, P, S)$. This means $T = X \cup \{a, b, c\}$. Let $\alpha, \beta\colon T^* \to \mathbb{Z}$ be the morphisms with

$$\alpha(w) = |w|_a - |w|_b, \qquad \beta(w) = |w|_b - |w|_c.$$

Then $\alpha(w) = \beta(w) = 0$ for each $w \in M \subseteq K$. Thus, Lemma 9.2.12 provides G-compatible extensions $\hat{\alpha}, \hat{\beta}\colon (N \cup T)^* \to \mathbb{Z}$ of α and β, respectively.

We define $k = \max\{|\hat{\alpha}(A)|, |\hat{\beta}(A)| \mid A \in N\} + 1$ and consider the $\mathcal{C}$-grammar $G' = (N, X, P', S)$, where $P' = \{A \to \pi_{N\cup X}(L) \mid A \to L \in P\}$. Then clearly $L(G') = \pi_X(M) = L$. We claim that G' is k-bursting. By Lemma 10.2.2, this implies $L = L(G') \in \mathcal{C}$ and hence the proposition.

Let t be a derivation tree for G' and x a node in t with $|\mathrm{yield}(x)| > k$. Then by definition of G', then there is a derivation tree $\bar{t}$ for G such that t is obtained from $\bar{t}$ by deleting or replacing by an ε-leaf each $\{a, b, c\}$-labeled leaf. Since x has to be an inner node, it has a corresponding node $\bar{x}$ in $\bar{t}$. Since G generates M, we have

$$\mathrm{yield}(\bar{t}) = a^n b x_1 b x_2 \cdots b x_n c^n$$

for some $n \geqslant 0$ and $x_1, \ldots, x_n \in X$, $x_1 \cdots x_n \in L$. Moreover, $\mathrm{yield}(\bar{x})$ is a factor of $\mathrm{yield}(\bar{t})$ and $\pi_X(\mathrm{yield}(\bar{x})) = \mathrm{yield}(x)$. This means $|\pi_X(\mathrm{yield}(\bar{x}))| > k$ and since in $\mathrm{yield}(\bar{t})$, between any two consecutive X-symbols, there is a b, this implies $|\mathrm{yield}(\bar{x})|_b > k - 1$. Let A be the label of x and $\bar{x}$. By the choice of k, we have $|\hat{\alpha}(\mathrm{yield}(\bar{x}))| = |\hat{\alpha}(A)| \leqslant k - 1$ and $|\hat{\beta}(\mathrm{yield}(\bar{x}))| = |\hat{\beta}(A)| \leqslant k - 1$. Hence, $|\mathrm{yield}(\bar{x})|_b > k - 1$ implies $|\mathrm{yield}(\bar{x})|_a \geqslant 1$ and $|\mathrm{yield}(\bar{x})|_c \geqslant 1$. However, a factor of $\mathrm{yield}(\bar{t})$ that contains an a and a c has to comprise all of $b x_1 \cdots b x_n$. Hence

$$\mathrm{yield}(x) = \pi_X(\mathrm{yield}(\bar{x})) = x_1 \cdots x_n = \pi_X(\mathrm{yield}(\bar{t})) = \mathrm{yield}(t).$$

This proves that G' is k-bursting. □

We are now ready to prove Theorem 10.1.4.

Proof of Theorem 10.1.4. First, note that if $V_i \in G_i \setminus F_i$, then $U_{i+1} \in F_{i+1} \setminus G_i$: By construction of U_{i+1}, the fact that $V_i \in G_i$ implies $U_{i+1} \in \mathrm{SLI}(G_i) = F_{i+1}$. By Proposition 2.7.2, F_i is a union closed full semi-trio. Thus, if we had $U_{i+1} \in G_i$, then $U_{i+1} \in \mathrm{Alg}(F_i)$ and Proposition 10.2.3 would imply $V_i \in F_i$, which is not the case.

Second, observe that $U_{i+1} \in F_{i+1} \setminus G_i$ implies $V_{i+1} \in G_{i+1} \setminus F_{i+1}$: By construction of V_{i+1}, the fact that $U_{i+1} \in F_{i+1}$ implies $V_{i+1} \in \mathrm{Alg}(F_{i+1})$, meaning $V_{i+1} \in G_{i+1}$. By Proposition 2.7.2, G_i is a full semi-AFL and according to Theorem 9.2.5, every language in G_i has a PAIM in G_i. Hence, if we had $V_{i+1} \in F_{i+1} = \mathrm{SLI}(G_i)$, then Proposition 10.2.1 would imply $U_{i+1} \in G_i$, which is not the case.

Hence, it remains to be shown that $V_0 \in G_0 \setminus F_0$. That, however, is clear because $V_0 = \#_0^*$, which is context-free and infinite. □

10.3 Conclusion

We have presented several results concerning strict inclusions between language classes induced by graph monoids. Most notably, we have shown that alternating the transformations *building stacks* and *adding blind counters* yields more languages in every step. This establishes some foundation to understand how far we have to nest the stacked counters to arrive at the behavior we want. The proof relies on two ingredients, the Parikh annotations with insertion marker from Chapter 9 and bursting grammars, which were introduced in this chapter.

The main result of this chapter, the strictness of the hierarchy, appeared in **[Zetzsche2015a]**.

Open problems

1. While the author strongly suspects that

$$\mathsf{VA}(M \times \mathbb{Z}^k) \subsetneq \mathsf{VA}(M \times \mathbb{Z}^{k+1}), \tag{10.5}$$
$$\mathsf{VA}(M \times \mathbb{Z}) \subsetneq \mathsf{VA}(\mathbb{B} * (M \times \mathbb{Z})) \tag{10.6}$$

   for each $M \in \mathsf{SC}^-$ and $k \in \mathbb{N}$, this does not seem to follow easily with the available proof techniques. Note that aside from Corollary 10.1.6, this would generalize every result in this chapter.

   The reason why the techniques do not provide these finer distinctions is twofold. First, in Section 9.2, Parikh annotations are only shown to exist on each level of the hierarchy. It is therefore conceivable that a PAIM for L might require more counters than L itself. This, in turn, calls into question whether there is an analog of Proposition 10.2.1 for constructing languages in the class $\mathsf{VA}(M \times \mathbb{Z}) \setminus \mathsf{VA}(M)$.

   Regarding (10.6), the problem is that the current separating languages require two extra counters for each level of the hierarchy. Thus, if M in (10.5) always only adds one counter before building another stack, the separating language may not be included in the right-hand side.

2. The problem of whether the inclusions $\mathsf{VA}(\mathbb{B}^k) \subseteq \mathsf{VA}(\mathbb{B}^{k+1})$ are strict remains open. Strictness has been conjectured by **Jantzen1979** **[Jantzen1979]**.

Related work As mentioned above, several non-expressibility results had been obtained before, either explicitly or implicitly. Proofs for the fact that $\mathsf{VA}(\mathbb{Z}^k)$ is strictly included in $\mathsf{VA}(\mathbb{Z}^{k+1})$ have been obtained by Latteux **[Latteux1977]**, **FernauStiebe2002a** **[FernauStiebe2002a]**, and by **ClearyElderOstheimer2006** **[ClearyElderOstheimer**
Vicolov1994 **[Vicolov1994]** and later again **FernauStiebe2002a** **[FernauStiebe2002a]** have shown that valence grammars over $\mathbb{Z}^k$ are less powerful than over $\mathbb{Z}^{k+1}$, which, together with an observation by **Hoogeboom2002** **[Hoogeboom2002]**, implies that $\mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{Z}^k)$ is strictly included in the class $\mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{Z}^{k+1})$.

Chapter 11

Language classes arising from valence automata

11.1 Introduction

Our aim is to use valence automata to obtain general insights on how structural properties of the storage mechanisms impact the computational properties of the result automata model. It is thus important to understand which storage mechanisms can can be captured by valence automata. Therefore, this chapter explores the limits of valence automata in terms of the language classes that arise from them.

The material is divided into two sections. In Section 11.2, we present necessary conditions that every language class $\mathsf{VA}(M)$ fulfills. In Section 11.3, we will investigate whether a common construction for storage mechanisms, *adding a zero-test*, has a counterpart for monoids. In other words, we ask: Given a monoid M, is there always a way to construct a monoid $\tilde{M}$ such that $\tilde{M}$ has the capabilities of M, plus a zero test?

11.2 Necessary conditions

In this section we present necessary conditions for a language class to arise from valence automata.

The first result has been obtained by **RenderKambites2010** [**RenderKambites2010**]. Its statement requires some terminology. Let M be a monoid. A subset $I \subseteq M$ is an *ideal* if $MIM \subseteq I$. It is called *proper* if it is non-empty and a strict subset. Moreover, an element $z \in M$ is called a *zero* if $xz = zx = x$ for every $x \in M$. Clearly, a monoid can have at most one zero, which is why one usually denotes this element by 0. A monoid is called *simple* if it has no proper ideals. Monoids with zero always have the ideal $\{0\}$, which is why they cannot be simple unless they are trivial. Therefore, we call them 0-simple if $\{0\}$ is their only proper ideal. **RenderKambites2010** have shown the following.

Theorem 11.2.1 (**RenderKambites2010 [RenderKambites2010]**). *For each monoid M, there is a simple or 0-simple monoid M' with $\mathsf{VA}(M) = \mathsf{VA}(M')$.*

Let us sketch the proof. Given an ideal I of a monoid M, we write $x \equiv_I y$ if $x, y \in I$ or $x = y$. Then, $\equiv_I$ is a congruence and the quotient monoid $M/\equiv_I$ is called *Rees quotient* and also denoted M/I. **RenderKambites2010** prove that for each proper ideal I, one has $\mathsf{VA}(M) = \mathsf{VA}(M/I)$. Then, Theorem 11.2.1 is shown as follows. If M is not already simple or 0-simple itself, then the union I of all proper ideals is proper, since none of the proper ideals can contain the identity. Furthermore, M/I is 0-simple and hence $M' = M/I$ is as required.

Using Theorem 11.2.1 and algebraic results on simple and 0-simple monoids, Render proved the following. Here, it can also be derived from Proposition 3.2.2 by noting that each group is either a torsion group or contains an isomorphic copy of $\mathbb{Z}$. Recall that a group G is called torsion group if for each $g \in G$, there is a $k \in \mathbb{N} \setminus \{0\}$ such that $g^k = 1$.

Theorem 11.2.2 (Render2010 [Render2010]). *For each monoid M, at least one of the following holds:*

1. $\mathsf{VA}(M)$ *contains the blind one-counter languages,*
2. $\mathsf{VA}(M)$ *contains the partially blind one-counter languages,*
3. $\mathsf{VA}(M) = \mathsf{VA}(G)$ *for some torsion group* G.

Note that none of the two conditions 1 and 2 implies the other since the classes $\mathsf{VA}(\mathbb{B})$ and $\mathsf{VA}(\mathbb{Z})$ are incomparable. The fact that $\mathsf{VA}(\mathbb{B})$ is not contained in $\mathsf{VA}(\mathbb{Z})$ follows by semilinearity arguments (see, for example, Corollary 7.1.4). On the other hand, Boasson **[Boasson1973]** proved hat $\mathsf{VA}(\mathbb{Z})$ is not contained in $\mathsf{VA}(\mathbb{B})$.

We use Theorem 11.2.2 to derive a more language-theoretic necessary condition for language classes $\mathsf{VA}(M)$. Since $\mathsf{VA}(G)$ is semilinear whenever G is a torsion group (Theorem 7.1.3), we immediately obtain the following.

Theorem 11.2.3. *For each monoid M, at least one of the following holds:*

1. $\mathsf{VA}(M)$ *contains the blind one-counter languages,*
2. $\mathsf{VA}(M)$ *contains the partially blind one-counter languages,*
3. $\mathsf{VA}(M)$ *is semilinear.*

As an application, we show that valence automata cannot realize automata with a counter that can only increase and checks in the end whether the counter value is a square. More precisely, we show that $\mathcal{T}(S)$, where $S = \{c^{n^2} \mid n \geqslant 0\}$, cannot be written as $\mathsf{VA}(M)$.

Corollary 11.2.4. *There is no monoid M with $\mathsf{VA}(M) = \mathcal{T}(S)$.*

Proof. Towards a contradiction, suppose $\mathsf{VA}(M) = \mathcal{T}(S)$. Then, since $\mathcal{T}(S)$ contains the non-semilinear language S, it has to contain either the blind one-counter languages, $\mathsf{VA}(\mathbb{Z})$, or the partially blind one-counter languages, $\mathsf{VA}(\mathbb{B})$. In any case, it contains $\{a^n b^n \mid n \geqslant 0\}$.

We show that this is not true by proving the following claim: For every $L \in \mathcal{T}(S)$, there is a constant k such that every $w \in L$ with $|w| \geqslant k$ decomposes as $w = xyz$ such that $|y| > 0$ and for some $m > 1$, we have $xy^m z \in L$. Clearly, the claim refutes that $\{a^n b^n \mid n \geqslant 0\}$ belongs to $\mathcal{T}(S)$ and thus completes the proof.

Suppose $L = TS$ for some rational transduction T. If k is larger than the number of states in a transducer for T, then each $w \in L$ with $|w| \geqslant k$ causes the transducer to repeat a state after reading a positive number of symbols. For the resulting decomposition $w = xyz$, we let ℓ be the number of c's the transducer reads while it outputs y and let n^2 be the number of c's it reads during the whole computation. In the case $\ell = 0$, we can clearly choose any $m > 1$, so suppose $\ell > 0$. Now observe that

$$n^2 + (2n + \ell)\ell = n^2 + 2n\ell + \ell^2 = (n + \ell)^2$$

is also a square, meaning $xy^{2n+\ell+1}z \in L$. Thus, with $m = 2n + \ell + 1$, the claim is satisfied. □

11.3 Zero tests

In order to model storage mechanisms using monoids, it is important to understand which transformations of storage mechanisms correspond to transformations of monoids. For example, taking the direct product $M \times N$ of monoids M and N corresponds to combining two storage mechanisms such that they can be used simultaneously and independently. Another example is taking the free product $\mathbb{B} * M$ with the bicyclic monoid: As explained in Section 2.4 (page 19), this corresponds to *building stacks* of configurations of an old storage mechanism.

Zero tests A very common transformation of storage mechanisms is the introduction of a *zero test*: One adds a new operation that succeeds only if the current configuration is empty. As an example, consider *one counter automata* [**Berstel1979**], in which one counter can be incremented, decremented, or tested for zero. Here, the counter cannot go below zero. Hence, these automata can be thought of as extending partially blind one counter automata with a zero test.

In this section, we are concerned with the question whether there is a transformation of monoids corresponding to adding zero tests. In other words, can we, for each monoid M, find a monoid M' that extends the storage mechanism represented by M with a zero test? Of course, we want M' to add precisely the capabilities of a zero test and nothing more. Otherwise, a trivial solution would be $\mathbb{B} * M$, since popping the stack always requires that the topmost entry is empty; but $\mathbb{B} * M$ will sometimes add more computational power than mere zero testing would: If $M = \mathbb{B}$, then valence automata over $\mathbb{B} * \mathbb{B}$ accept all context-free languages whereas one-counter automata (with zero test) do not (see, for example, [**Berstel1979**]).

Formalization Let us formalize our question. Suppose M is a finitely generated monoid with an identity language $L \subseteq X^*$. If we could find a monoid M' with identity language $(L\#)^*$ for some $\# \notin X$, this would certainly constitute a solution: Elements represented by X^* would correspond to elements in M and the zero test could be realized by multiplying the element represented by $\#$. However, it is easy to see that $(L\#)^*$ does not, in general, arise as an identity language: Suppose $\iota\colon X^* \to M$ is a surjective morphism and $\iota(u), \iota(v) \in M$ with $\iota(u)\iota(v) = 1$ and $\iota(v) \neq 1$. Then $uv\# \in (L\#)^*$ and if $(L\#)^*$ were an identity language, we would also have $u(uv\#)v\# \in (L\#)^*$, contradicting $\iota(v) \neq 1$.

While we cannot hope to construct a monoid with identity language $(L\#)^*$, we can describe the class of languages resulting from a storage mechanism consisting of "M plus zero test". It consists of those obtainable from $(L\#)^*$ by rational transductions: A finite state transducer with input language $(L\#)^*$ can be thought of as a finite automaton with access to the storage of M and a zero test. In other words, the language class is the principal full trio $\mathcal{T}((L\#)^*)$. We can therefore ask whether we can at least find a monoid that realizes "M plus zero test" with respect to the generated class of languages. In other words: Is there a monoid M' such that $\mathsf{VA}(M') = \mathcal{T}((L\#)^*)$? Unfortunately, the answer is negative again, as stated by the main result of this section.

Theorem 11.3.1. *Let $L \subseteq X^*$ be an identity language of a finitely generated infinite torsion group and $\# \notin X$. There is no monoid M with $\mathsf{VA}(M) = \mathcal{T}((L\#)^*)$.*

It should be mentioned that finitely generated infinite torsion groups indeed exist. Whether this is true had been a long-standing open question—known as the *Burnside problem*—before it was answered positively by **GolodShafarevich1964 [GolodShafarevich Golod1964]**. For simple constructions of such groups, see **[GuptaSidki1983, Grigorchuk1980]**.

We shall prove Theorem 11.3.1 by showing that the class $\mathcal{T}((L\#)^*)$ violates each of the three conditions of Theorem 11.2.2. If condition 1 or 2 is fulfilled, then $\mathsf{VA}(M)$ contains $\{a^m b^m \mid m \geqslant 0\}$. The first step in the proof of Theorem 11.3.1 is to show that $\mathcal{T}((L\#)^*)$ is too small to contain $\{a^m b^m \mid m \geqslant 0\}$.

Lemma 11.3.2. *Let L be an identity language of a finitely generated torsion group and let $K \in \mathcal{T}((L\#)^*)$. Then there is an $n \in \mathbb{N}$ such every $w \in K$ with $|w| \geqslant n$ decomposes into $w = xyz$ with $|y| \geqslant 1$ and $xy^t z \in K$ for infinitely many t. In particular, $\{a^m b^m \mid m \geqslant 0\}$ does not belong to $\mathcal{T}((L\#)^*)$.*

Proof. It clearly suffices to prove the first statement. Let $L \subseteq X^*$ be an identity language of G and let $\iota\colon X^* \to G$ be the surjective morphism inducing L. Moreover, let $Y = X \cup \{\#\}$ and $K = T(L\#)^*$, $K \subseteq Z^*$, for some rational transduction $T \subseteq Y^* \times Z^*$. Let T be given by the automaton $A = (Q, Y, Z, E, q_0, F)$. We may assume that

$$E \subseteq (Q \times Y \times \{\varepsilon\} \times Q) \cup (Q \times \{\varepsilon\} \times Z \times Q)$$

and that every edge entering a final state is labeled with $(\#, \varepsilon)$, that is, it reads # and outputs nothing. Moreover, let $k = |Q|$ and $n = k^2 + 1$. Consider a word $v \in K$ with $|v| \geqslant n$. Then

$$\begin{aligned}(q_0, (\varepsilon, \varepsilon)) &\to_A^* (q_1', (u_1, v_1)) \\ &\to_A (q_1, (u_1\#, v_1)) \\ &\vdots \\ &\to_A^* (q_m', (u_1\# \cdots u_m, v_1 \cdots v_m)) \\ &\to_A (q_m, (u_1\# \cdots u_m\#, v_1 \cdots v_m))\end{aligned}$$

for some $m \in \mathbb{N}$, $q_1', q_1, \ldots, q_m', q_m \in Q$, $u_1, \ldots, u_m \in X^*$, $v_1 \cdots v_m = v$, and $q_m \in F$. We distinguish two cases.

- Suppose $|v_i| > k$ for some $1 \leqslant i \leqslant m$. Then write $v_i = y_1 \cdots y_\ell$ for $y_1, \ldots, y_\ell \in Z$ and define $\hat{u} = u_1\# \cdots u_{i-1}\#$ and $\hat{v} = v_1 \cdots v_{i-1}$. Refin-

ing the run $(q_i, (\hat{u}, \hat{v})) \to_A^* (q'_{i+1}, (\hat{u}u_i, \hat{v}v_i))$ yields

$$\begin{aligned}(p_0, (\hat{u}, \hat{v})) &\to_A^* (p_1, (\hat{u}x_1, \hat{v}y_1)) \\ &\to_A^* (p_2, (\hat{u}x_1x_2, \hat{v}y_1y_2)) \\ &\vdots \\ &\to_A^* (p_\ell, (\hat{u}x_1 \cdots x_\ell, \hat{v}y_1 \cdots y_\ell))\end{aligned}$$

with $p_0, \ldots, p_\ell \in Q$ and $x_1, \ldots, x_\ell \in X^*$, where $p_0 = q_i$ and $p_\ell = q'_{i+1}$. Since $\ell > k = |Q|$, there are $1 \leqslant r < s \leqslant \ell$ with $p_r = p_s$. Since G is a torsion group, there are infinitely many $t \geqslant 1$ that satisfy the equality $\iota((x_{r+1} \cdots x_s)^t) = \iota(x_{r+1} \cdots x_s)$ and hence

$$v_1 \cdots v_{i-1}(y_1 \cdots y_r(y_{r+1} \cdots y_s)^t y_{s+1} \cdots y_\ell)v_{i+1} \cdots v_m \in K.$$

Since $|y_{r+1} \cdots y_s| = |s - r| \geqslant 1$, we have found the desired decomposition.

- Suppose $|v_i| \leqslant k$ for all $1 \leqslant i \leqslant m$. Consider the set $J = \{i \mid |v_i| \geqslant 1\}$. Since $\sum_{i \in J} |v_i| = |v| > k^2$, we have $|J| > k$. This means there are $i, j \in J$, $i < j$, with $q_i = q_j$. For each $t \geqslant 1$, we have

$$u_1\# \cdots u_i\#(u_{i+1}\# \cdots u_j\#)^t u_{j+1} \cdots u_m \in (L\#)^*.$$

Hence, the equality $q_i = q_j$ implies $v_1 \cdots v_i(v_{i+1} \cdots v_j)^t v_{j+1} \cdots v_m \in K$. Since $|v_{i+1} \cdots v_j| \geqslant |v_j| \geqslant 1$, we have found the desired decomposition.

□

Together with Theorem 11.2.2, the foregoing lemma implies that if $\mathcal{T}((L\#)^*)$ is of the form $VA(M)$ for some monoid M, then there is a torsion group G with $VA(G) = VA(M)$. The next step in our proof of Theorem 11.3.1 is to show that $\mathcal{T}((L\#)^*)$ is too large to be contained in $VA(G)$ for a torsion group G.

Lemma 11.3.3. *Let $L \subseteq X^*$ and $\# \notin X$. If $(L\#)^* \in VA(G)$ for a torsion group G, then L is regular.*

Note that Corollary 10.1.8 states that the same is true of groups $\mathbb{Z}^n$ (as opposed to torsion groups). However, Lemma 11.3.3 and Corollary 10.1.8 use quite different arguments.

Proof of Lemma 11.3.3. Towards a contradiction, suppose $(L\#)^* = L(A)$ for a valence automaton $A = (Q, X \cup \{\#\}, G, E, q_0, F)$. First, we define

$$\begin{aligned}R = \{p \in Q \mid\ &(q_0, \varepsilon, 1) \to_A^* (p, s, f) \to_A^* (r, st, 1) \\ &\text{for some } f \in G,\ s, t \in (X^*\#)^*, \text{ and } r \in F\}.\end{aligned}$$

In other words, R consists of those states that can be entered in a valid computation after reading a prefix in $(X^*\#)^*$. Moreover, for states $p, q \in R$, we write $p \rightsquigarrow q$ if $(p, \varepsilon, 1) \to_A^* (q, w, g)$ for some $w \in (X^*\#)^*$ and $g \in G$. With this, let $P = \{(p, q) \in R \times R \mid q \rightsquigarrow p\}$. Finally, for each $(p, q) \in P$, define $K_{p,q}$ to be the regular language

$$K_{p,q} = \{w \in X^* \mid (p, \varepsilon, 1) \to_A^* (q, w\#, g) \text{ for some } g \in G\}.$$

We claim that

$$L = \bigcup_{(p,q)\in P} K_{p,q} \tag{11.1}$$

which clearly implies the lemma.

We begin with the inclusion "$\subseteq$". Let $w \in L$ and $k = |Q|$. Since $(w\#)^{k+2}$ belongs to $(L\#)^*$, we have

$$\begin{aligned}(q_0, \varepsilon, 1) &\to_A^* (q_1, w\#, g_1)\\ &\to_A^* (q_2, (w\#)^2, g_1)\\ &\vdots\\ &\to_A^* (q_{k+1}, (w\#)^{k+1}, g_{k+1}) = (q_{k+1}, (w\#)^{k+1}, 1).\end{aligned}$$

Then $q_0, \ldots, q_{k+1} \in R$ and there are indices $1 \leqslant i < j \leqslant k+1$ such that $q_i = q_j$. Since $q_{i+1} \rightsquigarrow q_j = q_i$, we have $(q_i, q_{i+1}) \in P$. Since furthermore $w \in K_{q_i,q_{i+1}}$, the word w is contained in the right-hand side of (11.1). This proves "$\subseteq$" of (11.1).

Now suppose $u \in K_{p,q}$ with $p, q \in R$ and $q \rightsquigarrow p$. Then we have

$$(q_0, \varepsilon, 1) \to_A^* (p, s, f) \to_A^* (r, st, 1) \qquad \text{since } p \in R \tag{11.2}$$

$$(p, \varepsilon, 1) \to_A^* (q, u\#, g) \qquad \text{since } u \in K_{p,q} \tag{11.3}$$

$$(q, \varepsilon, 1) \to_A^* (p, v, h) \qquad \text{since } q \rightsquigarrow p \tag{11.4}$$

for some $g, h, f \in G$, $v, s, t \in (X^*\#)^*$, $r \in F$. Together, (11.3) and (11.4) tell us that

$$(p, \varepsilon, 1) \to_A^* (q, u\#, g) \to_A^* (p, u\#v, gh).$$

Since G is a torsion group, there is an $\ell \geqslant 1$ with $(gh)^\ell = 1$. Hence, repeating the former run yields

$$(p, \varepsilon, 1) \to_A^* (p, (u\#v)^\ell, (gh)^\ell) = (p, (u\#v)^\ell, 1).$$

By inserting this into (11.2), we can construct the run

$$\begin{aligned}(q_0, \varepsilon, 1) &\to_A^* (p, s, f)\\ &\to_A^* (p, s(u\#v)^\ell, f)\\ &\to_A^* (r, s(u\#v)^\ell t, 1),\end{aligned}$$

meaning $w = s(u\#v)^\ell t \in (L\#)^*$. Since $s \in (X^*\#)^*$ and $\ell \geqslant 1$, the word w starts with $u\#$ or has a factor $\#u\#$. In any case, $u \in L$. This proves (11.1) and thus the lemma. □

We are now ready to prove Theorem 11.3.1.

Proof of Theorem 11.3.1. Suppose $\mathcal{T}((L\#)^*) = \mathsf{VA}(M)$ for a monoid M. We distinguish the cases of Theorem 11.2.2.

If condition 1 or 2 is fulfilled, then $\{a^n b^n \mid n \geqslant 0\}$ is in $\mathsf{VA}(M)$. According to Lemma 11.3.2, this language does not belong to $\mathcal{T}((L\#)^*)$. Therefore, $\mathsf{VA}(M)$ has to satisfy condition 3 and we have $\mathcal{T}((L\#)^*) = \mathsf{VA}(H)$ for a torsion group H. This means in particular $(L\#)^* \in \mathsf{VA}(H)$, but then Lemma 11.3.3 implies that L is regular. However, L is the identity language of a finitely generated infinite group and hence non-regular by Theorem 3.3.1. □

11.4 Conclusion

We have investigated which languages classes arise from valence automata. Exploiting a result of **Render2010** [**Render2010**], we have obtained the following results. First, we have shown that every language class induced by valence automata contains $\mathsf{VA}(\mathbb{B})$, contains $\mathsf{VA}(\mathbb{Z})$, or is semilinear. We have applied this to a type of automata that has access to a counter (that can only increase) and accepts if the counter value is a square in the end. Our result implies that the class of languages accepted by such automata is not of the form $\mathsf{VA}(M)$.

We have also shown that there is no way to transform any monoid M into another monoid $\tilde{M}$ such that $\tilde{M}$ has the expressive power of M, plus zero tests.

Open problems

1. We have seen that it is not possible to turn each monoid M into a monoid M' such that M' represents the storage mechanism of M plus a zero test. The monoids M for which this turned out to be impossible are finitely generated infinite torsion groups. Since these seem to be of limited interest in the context of modeling infinite-state systems, one is inclined to ask whether adding zero tests is possible for a restricted class of monoids.

2. A particular case of the first problem is interesting in its own right: Does the class of languages accepted by one-counter automata with zero test arise as $\mathsf{VA}(M)$?

Related work The range of language classes that can be captured with monoid-defined storage mechanisms increases when one considers valence automata with target sets. These have been considered by **RedkoLisovik1980** [**RedkoLisovik1980**], **FernauStiebe2001** [**FernauStiebe2001**], and **RenderKambites2009** [**RenderKambites2009**]. Here, one specifies a subset of the storage monoid that has to be reached in order to accept. It should be noted that allowing arbitrary target sets leads very quickly to extremely powerful models. For example, with target sets and the free monoid as storage, one can accept any language. Therefore, the considered target sets are usually restricted to rational subsets of the storage monoid.

Another variant has been suggested later by **RenderKambites2010** [**RenderKambites2010**]. When one has target sets, it is not necessary to require the storage to be given by a monoid and it makes sense to consider arbitrary semigroups (which does not necessarily have an identity) instead. Once the requirement for an identity is dropped, allowing initial sets potentially increases the range of appearing language classes further.

Acknowledgements I would like to thank Dietrich Kuske for asking the question of the second open problem.

Chapter 12

Conclusion

We have studied various facets of expressiveness and analyzability of automata with storage. Specifically, for a series of computational properties, we have identified ways in which the structure of a storage mechanism impacts its computational properties. These investigations have been pursued using the formal framework of valence automata, in most cases over graph monoids. Since each of these properties is assigned its own chapter with its own conclusion section, we refer the reader to those sections for the individual properties.

On the whole, we can say that valence automata, especially over graph monoids, permit meaningful characterizations of computational properties of storage mechanisms. Furthermore, aside from the particular results that were obtained, the research on these models yielded new models that raise interesting questions.

New models For example, the results in Chapter 4 leave us with an interesting extension of the open problem of whether reachability is decidable for Petri nets with one pushdown storage: The extension starts from partially blind counters and allows building stacks and adding partially blind counters. The results here suggest that reachability might even be decidable even for this general model, which puts the problem for Petri nets with a pushdown in a new context.

There is another model that emerged from the foregoing investigations. In the course of this work, we have identified stacked counter automata as a model that is, on the one hand, rather expressive: Stacked counters are expressively complete among those storages that guarantee semilinearity. On the other hand, it seems to be well-suited for various kinds of analysis. We have seen that they guarantee semilinearity (Chapter 7), permit the removal of ε-transitions (Chapter 8), and allow the computation of downward closures (Chapter 9). It would therefore be interesting to explore *applications*.

Applications In the case that one has just a pushdown store and blind counters, an equivalent model has been studied and applied by **HagueLin2011** [**HagueLin2011**]. While their counters permit zero tests, they have to be reversal bounded, which makes them interchangeable with blind counters. This simple case of stacked counter automata corresponds to recursive programs with access to a set of (unbounded) global numeric variables. In more complex configurations, when we also build stacks, stacked counter automata can also model variables that are con-

fined to the scope of the current execution instance. Interestingly, while **HagueLin2011** show that reachability is NP-complete (for a fixed number of reversals) for their model, Theorem 8.1.1 means that at the least the membership problem remains in NP (albeit perhaps not uniformly) even when we use the full power of stacked counter automata.

Another source of applications of stacked counter automata is group theory. As already mentioned in Section 8.6, further insights on the uniform complexity of their membership problem would entail complexity bounds for the rational subset membership problem of graph groups.

These potential applications also call for the investigation of decidability and complexity of further problems. For example, extending the popular approach to LTL model checking of Büchi pushdown systems by **BouajjaniEsparzaMaler1997** [**BouajjaniEsparzaM** to some form of Büchi stacked counter systems seems promising. Another interesting problem to study would be the regularity problem for deterministic variants. It asks whether the accepted language of an automaton is regular. This is known to be decidable for deterministic pushdown automata [**Stearns1967**] and for deterministic blind multicounter automata (see Section 9.3.2). This suggests that the same could be true of a deterministic variant of stacked blind counter automata.

List of symbols

Monoids

$\mathbf{1}$	trivial monoid. 13
$\mathbb{B}$	bicyclic monoid. 14
$\mathbb{T}(X, I)$	traces over X with independence relation I. 98
$M * N$	free product. 13
$M *_F N$	free product with amalgamation. 27
$M^{(n)}$	n-fold free product of M. 14
M^n	n-fold direct product of M. 13
$\mathbb{M}(\Gamma, (M_v)_{v \in V})$	graph product. 75
$M\restriction_\Delta$	graph product, restricted to subgraph. 76
$\mathbb{M}\Gamma$	graph monoid defined by Γ. 20
$\langle S \rangle$	submonoid generated by S if $S \subseteq M$ for a monoid M. 7
S^*	submonoid generated by S if $S \subseteq M$ for a monoid M. 8
$S^\oplus$	submonoid generated by S if $S \subseteq C$ for a commutative monoid C. 8

Subsets of monoids

$R_1(M)$	right-invertible elements of M. 13
$L_1(M)$	left-invertible elements of M. 13
$H_1(M)$	invertible elements of M. 13
$J_1(M)$	elements $b \in M$ such that $abc = 1$ for some $a, c \in M$. 13
$\overrightarrow{1}(x)$	right inverses of x. 43
$\overleftarrow{1}(x)$	left inverses of x. 43

Classes of monoids

SL	class of monoids with $\mathbf{1}, \mathbb{B} \in \mathsf{SL}$ and where $M, N \in \mathsf{SL}$ implies $M * N \in \mathsf{SL}$, $M \times \mathbb{Z} \in \mathsf{SL}$. 84
DEC	class of monoids with $\mathbb{B}^n \in \mathsf{DEC}$ and where $M, N \in \mathsf{DEC}$ implies $M * N, M \times \mathbb{Z} \in \mathsf{DEC}$. 58
REM	class of monoids $\mathbb{M}\Gamma$ where Γ^- does not contain C_4 or P_4, but Γ contains a PPN-graph. 59
SC^-	class of monoids with $\mathbf{1} \in \mathsf{SC}^-$ and where $M \in \mathsf{SC}^-$ implies $\mathbb{B} * M, M \times \mathbb{Z} \in \mathsf{SC}^-$. 84
$\mathsf{SC}^\pm$	class of monoids with $\mathbb{B}^n \in \mathsf{SC}^\pm$ and where $M \in \mathsf{SC}^\pm$ implies $\mathbb{B} * M, M \times \mathbb{Z} \in \mathsf{SC}^\pm$. 59
SC^+	class of monoids with $(\mathbb{B} * \mathbb{B}^n) \times \mathbb{B} \in \mathsf{SC}^+$ and where $M \in \mathsf{SC}^+$ implies $\mathbb{B} * M, M \times \mathbb{B} \in \mathsf{SC}^+$. 59

Languages

D_n	Dyck language (over n pairs of parentheses). 12

D'_n	semi-Dyck language (over n pairs of parentheses). 12
$L \sqcup\!\sqcup K$	shuffle product of L and K. 8
$L{\downarrow}$	downward closure of L. 33
$L{\uparrow}$	upward closure of L. 33
$\mathsf{SF}(G)$	sentential forms of $\mathcal{C}$-grammar G. 25

Functions and relations

$\preceq$	subword ordering. 34
$\preceq_\diamond$	insertion at marker. 126
$\leqslant_k$	component-wise $\leqslant$; k divides differences. 88
$\equiv_\Gamma$	congruence induced by graph Γ. 20
$\equiv_k$	component-wise congruent modulo k. 88
$\Psi(w)$	Parikh image of w. 8
$\pi_Y(w)$	projection of w onto subalphabet Y. 8
$\rho(w, U)$	number of U-factors in U-decomposition of w. 45
$\mathsf{dep}(w)$	dependence graph corresponding to w. 98
$[w]_\Gamma$	congruence class induced by graph Γ. 20
$[w]_I$	trace induced by w with independence relation I. 98
$[w]_{\Gamma\mid\mathbb{T}}$	trace induced by w with independence relation given by graph Γ. 98
$\mathrm{bin}(w)$	number obtained by interpreting w as binary representation. 87

Language classes

Reg	regular languages. 9
CF	context-free languages. 11
RE	recursively enumerable languages. 54
Σ_n	n-th level of the arithmetical hierarchy. 66
$\Sigma_n(L)$	n-th level of the arithmetical hierarchy relative to L. 66
Prio	languages accepted by priority multicounter machines. 61
$\mathsf{VA}(M)$	languages accepted by valence automata over M. 15
$\mathsf{VA}^+(M)$	languages accepted by ε-free valence automata over M. 93
$\mathsf{detVA}(M)$	languages accepted by deterministic valence automata over M. 41
$\mathsf{VG}(M)$	languages generated by valence grammars over M. 40
$\mathcal{T}(L)$	full trio generated by L. 10
$\mathcal{BT}(L)$	Boolean closed full trio generated by L. 69
$\mathsf{SLI}(\mathcal{C})$	languages $h(L \cap \Psi^{-1}(S))$ with h morphism, $L \in \mathcal{C}$, and S semilinear. 23
$\mathsf{Alg}(\mathcal{C})$	algebraic extension of language class $\mathcal{C}$. 25
F	union of all levels of the hierarchy F. 33
F_i	level of hierarchy F. 33
G_i	level of hierarchy F. 33

Graphs

Γ^-	Γ's underlying simple graph. 20
P_4	path on four vertices. 20
C_4	cycle on four vertices. 20

Transduction classes

$\mathsf{VT}(M)$	transductions performed by valence transducers over M. 41
$\mathsf{VT}(M, C)$	transductions performed by valence transducers over M with output in C. 104

$\mathsf{VT}^+(M, C)$	transductions performed by ε-free valence transducers over M with output in C. 104

Miscellaneous

X_Γ	alphabet $\{a_v, \bar{a}_v \mid v \in V\}$ for graph $\Gamma = (V, E)$. 20
$\mathbb{N}$	set of natural numbers. 7
$\mathbb{Z}$	set of integers. 7
$M \hookrightarrow N$	there is a morphism $\varphi\colon M \to N$ with $\varphi^{-1}(1) = \{1\}$. 28
$\mathsf{Rat}(M)$	set of rational subsets of M. 9
$\mathsf{SL}(M)$	set of semilinear subsets of commutative monoid M. 9
$X^\oplus$	multisets over X if X is an alphabet. 8
X^*	words over X if X is an alphabet. 8
ε	empty word. 8
$\mathcal{P}(A)$	power set of A. 7

Education

Feb. 2011 – June 2015	Doctoral student at TU Kaiserslautern.
Oct. 2003 – Dec. 2010	Studies in Computer Science at Universität Hamburg, with a minor in Mathematics. Degree: *Diplom-Informatiker* ("excellent")
Aug. 2000 – June 2003	Abitur (High School) at Berufsbildende Schulen Winsen/Luhe.