Higher Dimensional Automata

Ph.D. Thesis

by

Zoltán L. Németh

Supervisor: Prof. Zoltán Ésik

University of Szeged
Department of Foundations of
Computer Science

2007

Acknowledgments

First of all, I would like to express my deep gratitude to my supervisor Professor Zoltán Ésik for all the guidance and encouragement he has offered me throughout the period of this research. He first suggested the topic of higher dimensional languages, and guided my first, not-too-easy steps in this direction. Without his advice and unique support this thesis would never have become a reality.

I would also like to thank my teachers at the University of Debrecen, where I graduated, especially Prof. Pál Dömösi, who made me see the beauty of the theory of automata and formal languages and provided the opportunity for me to take part in my first AFL conference.

Other special thanks go to my colleagues at the Department of Computer Science in Szeged for helping to provide a pleasant working atmosphere. I especially grateful to Zsolt Gazdag, who gave me the LaTeX source of his thesis. In addition, I would like to thank David Curley for checking the thesis from a linguistic point of view and for providing some inspirational ideas during the writing stage.

Lastly, this list of acknowledgements would be incomplete without mentioning my parents, parents-in-law, and close friends who have always given me tremendous support and encouragement. And, of course, I wish to express my heart-felt thanks to my wife Sugárka and my sons Ábel, Regő and Nándor, for their patience and understanding over the past few years.

Contents

Chapter 1

Introduction

1.1 On the General Concept of Regularity

Finite automata together with regular languages form one of the cornerstones of theoretical computer science. They are actively investigated mathematical objects which are of unquestionable importance both from a theoretical and practical point of view [RS97]. Their widespread use is mainly due to two facts. First, words can serve as models for a wide range of sequential systems, as they can simulate sequential behavior quite naturally. Second, the concept of regular word languages can be defined in several different, but equivalent ways. To establish the terms and notations of the concepts which we will work with, we will fix the following terminology:

- ***Regularity*** will mean acceptance by finite automata.
- ***Recognizability*** will mean algebraic recognizability by finite algebras or finite-index congruences.
- ***Rationality*** will mean expressibility by rational (also called regular) expressions.
- ***MSO-definability*** will mean definability by monadic second-order logical formulas.

The names for these concepts reflect the names used by a French school on formal languages. But considering that these concepts are equivalent, and hence that they are freely interchangeable, some authors give them different meanings. For example it is common to speak about recognizability by automata, or to talk about regular expressions instead of rational ones.

In the following we will use these concepts not just for word languages, but also for languages of other structures. In addition, we shall employ the following notations for the corresponding language classes: Reg, Rec, Rat and MSO. The classical results

of automata theory (due to Büchi, Kleene, Myhill and Nerode) demonstrate that the equalities $\mathsf{Reg} = \mathsf{Rec} = \mathsf{Rat} = \mathsf{MSO}$ hold for languages of finite words.

It should be emphasized here that these four concepts are not simply four different ways of defining the same class of word languages, but rather each of them contains the essence of this class from a different perspective. In certain situations one of them may have some advantage and be better suited than the other three. E.g. the closure under homomorphism directly follows from the concept rationality, while closure under inverse homomorphisms is easily implied by recognizability. In model checking abstract specifications are frequently written in logical formulas, while in pattern matching rational expressions are quite useful.

Of course, there are many other computational models that have more complex structures than finite words. These include infinite words [PP04, Wil94], trees [GS84], traces [DR95], partially ordered sets (posets for short) [Pra86, LW98, LW00, Kus03a], message sequence charts [Kus03b] and graphs [Cou91, CW05]. These models were introduced and applied to capture such computational aspects like timing and concurrency.

When investigating these more complex models the natural question arises – which is of crucial importance – about what results of the classical theory of words can be generalized and how. In many important cases the above notions can be suitably defined and are known to be equivalent. But sometimes we are faced with serious problems. It is not always clear how to choose an appropriate algebraic or logical framework. And, for instance, for graphs, for posets, and even for sp-posets in general, the concept of an automaton that captures recognizability is not known. For a general overview of this topic, we refer to the paper by Weil [Wei04a] which surveys the concept of recognizability in computer science.

The subject of this thesis is about the generalization of the fundamental results of classical automata theory to higher dimensions. Both finite and infinite higher dimensional words and their languages will be defined and investigated.

Fortunately, we can restrict our studies to just the two-dimensional case, since both our concept and results can be readily generalized to any finite number of dimensions.

1.2 Two-dimensional Words (Biwords)

The reader may recall that a *monoid* is a set equipped with an associative operation which has an identity element. Next, let us fix an alphabet Σ. As usual, Σ^* denotes the *free monoid* over Σ. The elements of Σ^* are called *words*, while the subsets of Σ^* are called *languages*.

In a similar way, a *binoid* is a set equipped with two associative operations and these two operations share a common identity element. After we can consider the *free binoid* over Σ, which we will be denoted by $\Sigma^*(\bullet, \circ)$. This is well-defined by universal algebraic considerations. In the following the elements of $\Sigma^*(\bullet, \circ)$ will be called *biwords*, while the subsets of $\Sigma^*(\bullet, \circ)$ will be *binoid languages* (over Σ). It is natural to describe biwords by terms using the letters of Σ, parentheses and two operation symbols, but we will also find that biwords can be represented in several other equivalent ways. First, we only consider perhaps the most intuitive one of them, which will be called *two-dimensional words*.

To construct two-dimensional words from the letters of Σ, we need two independent concatenation operations. The first one will be called the *horizontal concatenation* (denoted by $\bullet$), while the second one will be called the *vertical concatenation* (denoted by $\circ$).

We will build two-dimensional words inductively from smaller elements called blocks. Initially we can use just the letters of Σ as blocks, then we can form more complex blocks by using the two concatenation operations. Naturally, the horizontal concatenation places some finite number of blocks to the left/right of each other, while the vertical concatenation places the blocks above/beneath each other. Now two-dimensional words are defined as those blocks that can be obtained from the elements of Σ by a finite number of applications of the two concatenations. We also have an *empty two-dimensional word* ε, which has no letters. The two-dimensional word which can be given by the term $a \bullet \langle b \circ \langle c \bullet d \rangle \rangle \bullet \langle e \circ f \rangle$ is illustrated in Figure 1.1.

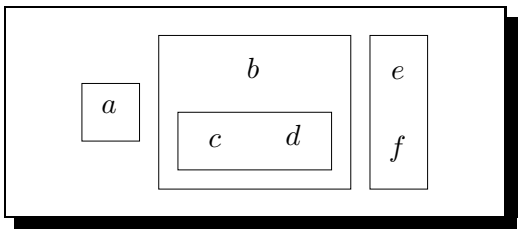

Figure 1.1: *The two-dimensional word* $a \bullet \langle b \circ \langle c \bullet d \rangle \rangle \bullet \langle e \circ f \rangle$.

Evidently the binary version of both concatenations is an associative operation on the set of two-dimensional words. Moreover, the empty two-dimensional word serves as an identity element for both operations. It is not hard to see that the algebra of two-dimensional words over an alphabet Σ is isomorphic to the free binoid generated by Σ in the variety of all binoids.

It is generally agreed that automata models operate on elements of some free algebra.

Classical finite automata operate on words, i.e. on elements of the free monoids (or on the elements of the free semigroups if we consider just nonempty words.) Tree automata operate on trees, which are the free Σ-algebras, and so on. Thus, if we want to generalize the notion of automata to higher dimensions, it is natural to examine how they operate on biwords, i.e. on the elements of the free binoids.

1.3 The Main Results of the Thesis

In the following we shall investigate the possibility of the extension of the four basic notions (namely recognizability, logical definability, regularity and rationality) to binoid languages.

Fortunately, one of the four concepts, namely *recognizability*, is uniquely determined by the fact that biwords (and hence binoid languages as well) are naturally equipped with two associative operations. Thus binoids are the only obvious choice for the algebraic framework for binoid languages.

Formulating the concept of *logical definability* is not as straightforward as recognizability, but it can be done with the help of a free algebra theorem presented in [Ési00] concerning so-called (m, n)-semigroups. Here (m, n)-semigroups are algebras, where m associative plus n associative and commutative operations are defined. Applying this theorem to the special case of binoids (i.e. when $m = 2$ and $n = 0$), we get a description of nonempty biwords (considered as elements of the free bisemigroups) by labeled sp-biposets.

In fact *biposets* are relational structures of the form $(P, <_h, <_v)$, where $<_h$ and $<_v$ are arbitrary partial order relations on the set P. If we add a labeling function $\lambda : P \to \Sigma$, where Σ is an alphabet, then $(P, <_h, <_v, \lambda)$ is a *labeled biposet*. Now the fact that biposets, and also their special cases – sp-biposets – are relational structures, allows us to interpret logical formulas on them, as in biposets the horizontal and vertical order relations are explicitly present. By doing so we obtain the concept of MSO-definability.

For the concept of *regularity* we introduce a new automata model called parenthesizing automata (PA for short). This model is one of the main contributions of the thesis. The name "parenthesizing" comes from the realization that a PA can be viewed as the union of two ordinary finite automata, where one component handles the horizontal concatenation of the subbiwords while the other handles their vertical concatenation. In addition, the two components are coupled by special, so-called parenthesizing transitions, which are labeled by parenthesis symbols. (See Definition 3.1 for more details.)

Next, we consider several *rational* classes of binoid languages, whose definitions

depend on what operations are allowed from the following list: Boolean operations (union, intersection, complementation), horizontal product ($\bullet$), vertical product ($\circ$), horizontal iteration ($*_\bullet$) and vertical iteration ($*_\circ$). Let $\mathsf{Fin}[\mathrm{op}_1, \ldots, \mathrm{op}_n]$ denote the class of those binoid languages that can be generated from the finite binoid languages by a finite number of applications of the operations op_1,...,op_n. In the thesis the following classes will be defined

- $\mathsf{HRat} = \mathsf{Fin}[\cup, \bullet, *_\bullet, \circ]$ the *horizontal rational languages*,
- $\mathsf{VRat} = \mathsf{Fin}[\cup, \circ, *_\circ, \bullet]$ the *vertical rational languages*,
- $\mathsf{BRat} = \mathsf{Fin}[\cup, \bullet, *_\bullet, \circ, *_\circ]$, the *birational languages*,
- $\mathsf{GRat} = \mathsf{Fin}[\cup, \bullet, *_\bullet, \circ, *_\circ, {}^-]$, the *generalized birational languages*, where $^-$, is the operation of taking the complement.

As usual, a binoid language is called *finite* if it contains a finite number of biwords. Similarly, a binoid language $L \subseteq \Sigma^*(\bullet, \circ)$ is *cofinite* if its complementer with respect to $\Sigma^*(\bullet, \circ)$ is finite. We denote the class of *finite languages* by Fin.

It is usual, and sometimes even necessary, to apply some restrictions on the structures being studied. These restrictions are sometimes naturally arise due to practical limitations – e.g. the finite number of the available processors. Here we will study three such restricted classes of binoid languages: HB – the class of horizontally bounded languages, VB – the class of vertically bounded languages, and BD – the class of bounded (alternation) depth languages.

As for the definitions, the easiest way to define horizontally and vertically bounded binoid languages is through their sp-biposet representations. Recall that a *chain* in a poset is a subset in which each pair of elements is comparable, i.e. a totally ordered subset. The *height* of a poset is the cardinality of a longest (maximum cardinality) chain. If $(P, <_h, <_v, \lambda)$ is a biposet, let its *horizontal height* be the height of the poset $(P, <_h)$. Similarly let its *vertical height* be the height of the poset $(P, <_v)$. Now a binoid language is *horizontally* (resp. *vertically*) *bounded* if there is an upper bound for the horizontal (resp. vertical) height of the sp-biposet representations of its elements.

We say that a binoid language L has a *bounded depth* if there is an integer K such that, for every biword $w \in L$, the maximal number of nested parentheses in the term representation of w is at most K. Let BD denote the class of binoid languages that have a bounded depth.

We established the inclusion relations among the considered classes. These can be summarized in Figure 1.2. First of all the classes of recognizable, regular and MSO-definable languages coincide. The class GRat of generalized birational binoid languages

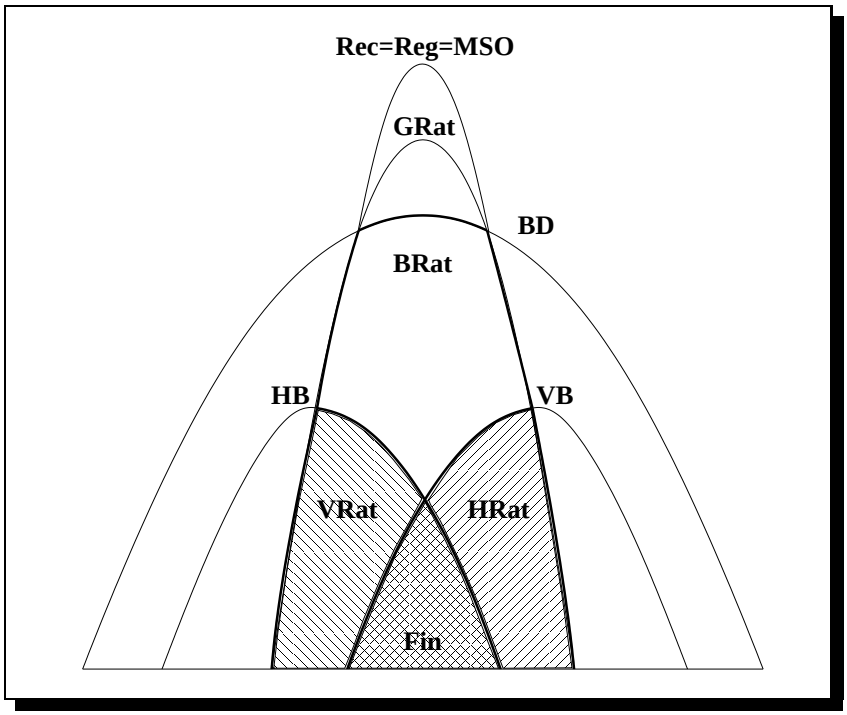

Figure 1.2: *Comparison of language classes of finite biwords.*

is strictly included in this class, which in turn strictly includes the class BRat of birational languages. The class BRat can be characterized as those regular binoid languages that have bounded depth, and the class VRat as those that are horizontally bounded. Similarly HRat is the intersection of the classes of regular (Reg) and vertically bounded languages (VB). From the definitions it directly follows that the intersection of HB and VB is the class of finite languages (Fin). Moreover, all inclusions suggested by the figure are strict.

An important feature of parenthesizing automata that an automata may possess any finite number of parenthesis pairs. The question emerges naturally if this feature is really necessary, or the number of parentheses can be bounded. In other words, we want to know whether there is a number K such that each regular binoid language can be accepted by a parenthesizing automaton with at most K pairs of parentheses. Theorem 3.32 will supply the answer to this question. We will show that no such K exists, i.e. there is no upper bound. Furthermore, if Reg_m denotes the class of all regular binoid languages that can be accepted by an automaton with $m \geq 0$ pairs of parentheses, then the classes $\mathsf{Reg}_0 \subsetneq \mathsf{Reg}_1 \subsetneq \mathsf{Reg}_2 \subsetneq \ldots$ form a strict hierarchy. And we will prove that this hierarchy is proper even when we consider languages over any

fixed alphabet Σ (Theorem 3.33).

An independent study on automata and regularity of binoid languages was done by Hashiguchi et al., and their results were presented in a series of papers [HIJ00, HWJ03, HSJ04]. They applied ordinary finite automata operating on term representation of biwords to define two classes of regular binoid languages. See the next section for more details. The aim of Section 3.12 is to relate this approach to our theory of parenthesizing automata. We will see that the classes of regular languages defined in [HIJ00] are strictly included in Reg. Moreover, we will show how the monoid approach of Hashiguchi et al. can be extended to our (boarder) class of regular binoid languages. This means that with appropriate definitions monoid automata are also capable of capturing the same concept of regularity. This will give a fourth equivalent characterization of the class Reg.

In Chapter 4 we will extend our investigations to infinite biwords. First we will define ω-bisemigroups in the pattern of ω-semigroups related to infinite words [PP04]. Now ω-biwords as abstract objects are just the elements of the free ω-bisemigroups. Similarly to the finite case, we can represent ω-biwords by certain infinite biposets. Next, we will present a graph-theoretic characterization of these biposets. Afterwards, we will examine the tree and term representations of ω-biwords. It will be followed by the extension of recognizability, MSO-definability and regularity to ω-binoid languages. To extend regularity we will also need to define the concept and the operation of parenthesizing Büchi-automata. Lastly we will prove that with the new concept of parenthesizing Büchi-automaton the equivalence of regularity, recognizability and MSO-definability remains true.

In the thesis we mostly concentrate on the general theory of regularity, but we believe that the concept of binoid languages is sufficiently general to have some practical applications as well. The reader can peruse the study by Hashiguchi et al. on bicodes [HKJ02] and on a modified RSA cryptosystem based on bicodes [HHJ03]. In the future biwords may also be used in modeling systems like sp-posets, which often serve as models for the behavior of modularly constructed concurrent systems (cf. [Pra86]). It would also be good to look for other concrete applications of our theory. Since the special feature of biwords and their n-dimensional generalizations is that they are naturally equipped with some *nested structures*, it seems obvious to look for applications where some nestedness (of arbitrary depth) is present, e.g. in XML databases and in modeling recursive function calls.

1.4 Related Work

The studies which are relevant to our investigations can be divided into two groups. First there are some papers that deal with the same thing, namely that of binoid languages. This group includes the papers of Hashiguchi et al. [HIJ00, HWJ03, HSJ04] and Dolinka [Dol05, Dol07]. Second, our concepts and methods are closely related to that of other generalizations of automata and language theory, like those presented in papers on infinite words, trees, posets, texts, traces, event structures, message sequence charts and graphs. Since the literature for this is quite extensive we shall only mention those which had the greatest influence on our work. For a more detailed and complete overview please consult with [Wei04a] and [RS97].

1.4.1 (m,n)-structures

One of our starting points will be the concept of (m, n)-structures introduced by Ésik in [Ési00], where m and n are nonnegative integers. They are a common generalization of both posets and component reducible graphs (or cographs, [CLB81]), and form the free algebras in the variety, where m associative plus n associative and commutative operations are defined. These will be introduced in Section 2.4. We will also discuss some of their special cases, of which the most important for us is that of sp-biposets. This is because sp-biposets are a possible representation of biwords and they are essential for the logical definability of binoid languages.

1.4.2 Poset Languages

Our investigations have been influenced to a great extent by the work of Lodaya and Weil [LW98, LW00] and Kuske [Kus01, Kus03a] on series-parallel posets languages and *branching automata* accepting them. These posets are also considered as a possible model for the behavior of modularly constructed concurrent systems [Pra86, Nie01].

Posets are naturally equipped with two operations called *series product* and *parallel product* [LW98]. In fact the parallel product is just disjoint union, and hence commutative. *Series-parallel posets* (or just sp-posets) are those posets that can be generated from the singletons by the two operations. It is known that sp-posets represent the elements of the free "semi-commutative bisemigroups", i.e. of the algebras equipped with an associative and an associative and commutative operation [Gis88]. These algebras are also called *series-parallel algebras* (or sp-algebras) [LW98, LW00]. Thus, sp-posets languages can be regarded as a two-dimensional generalization of the classical theory

to a situation where we have two associative operations, but one of them is also commutative. Also, sp-posets may be characterized as those posets that does not contain an induced subgraph isomorphic to the "N" directed graph [Gra81].

Gischer [Gis88] studied the equational theory of sp-posets and sp-poset languages. In [LW98, LW00], Lodaya and Weil defined recognizable languages of sp-posets as well as regular languages accepted by branching automata, and rational languages. They showed that a language of sp-posets is regular iff it is rational, and that the recognizable languages form a proper subclass of the regular languages. Aside from semi-commutativity, their notion of recognizability corresponds to ours. On the other hand, their notion of rationality is much more general than our notion of birationality, and although our parenthesizing automata owe much to their branching automata, our version is not a non-commutative version of branching automata. The above differences, together with the well-known fact that rationality and recognizability are not equivalent for free commutative semigroups, help explain why the above-mentioned results of Lodaya and Weil seem so different from ours.

Nevertheless, Lodaya and Weil also obtained several results that are similar to ours. They studied *bounded-width* poset languages that correspond to our vertically bounded binoid languages (VB) and showed in [LW00] that for such languages, the concepts of recognizability, regularity and series rationality are all equivalent. Moreover, Kuske proved in [Kus03a] that for bounded width poset languages, these conditions are equivalent to MSO-definability. These equivalences correspond to our Corollary 3.71, the vertically bounded case.

What we called a birational binoid language corresponds to the series-parallel rational sp-poset languages of Lodaya and Weil. In [Kus01, Kus03a], Kuske proved that any series rational poset language is MSO-definable and that every MSO-definable poset language is recognizable. On the other hand, there easily exist recognizable but not MSO-definable sp-poset languages.

The main object of study in [LW01] is the extension of the classical framework to automata over free algebras with a single associative operation and a collection of operations not satisfying any nontrivial equations. It is shown that a suitably adapted version of branching automata captures recognizable languages, and that there exists a corresponding notion of rationality. Lodaya and Weil also discussed, albeit in a rather indirect way, the situation when at least one of the additional operations is associative. In this case they found that the recognizable languages form a proper subclass of the regular languages that coincide with the rational languages. Their "asymmetric" notion

of regularity is different from ours (which is "symmetric"), and their notion of rationality (which they showed to correspond to regularity) is much more general than ours. Our Proposition 3.52 is also found in [LW01].

A graph-theoretic characterization and a free algebra theorem of the posets that can be constructed from the singletons by series and parallel products and by series ω-power can be found in [BÉ98], while [ÉO99] discusses the case when a parallel ω-power is also allowed.

1.4.3 2-structures and Text Languages

Our investigation also owes much to the work of Hoogeboom and ten Pas [HtP96, HtP97] on *text languages*. In Section 3.11 we will use their result that establishes the equality $\mathsf{Rec} = \mathsf{MSO}$ for text languages in order to show that the same equality holds for binoid languages as well.

Historically, text languages arose as special cases of 2-structures. 2-structures are relational structures, and can actually be viewed as generalizations of graphs. They were introduced to provide a convenient general framework for the investigation of decompositional and transformational properties of graphs (and graph-like structures). From this motivation 2-structures have been studied extensively and many results for them are known. The interested reader can consult with the book edited by Ehrenfeucht and Rozenberg [ER90].

As was mentioned previously, texts are special cases of 2-structures, but a text can also be defined as a labeled set equipped with two linear order relations, or equivalently as a word with an additional linear order, see Definition 3.62.

In Section 3.11 we shall find that biwords – more precisely the sp-biposet representations of biwords – can be identified with some special cases of texts. These texts are called *alternating* or *uniformly nonprimitive texts*. See [EHPR96] about uniformly nonprimitive 2-strtuctures. We will describe the correspondence of sp-biposets and alternating texts in detail in Section 3.11.3.

1.4.4 Finite Automata on Binoid Languages

Automata and languages over free binoids have also been studied independently of us by Hashiguchi (with various coauthors) in a series of papers [HIJ00, HWJ03, HSJ04]. In these, the authors consider biwords in their *term representation*[1]. Here, of course, term representations mean words over the extended alphabet $E(\Sigma) = \Sigma \cup \{\langle, \rangle, \bullet, \circ\}$,

[1] In [HIJ00, HWJ03, HSJ04], term representations are called *s-forms*.

where $\langle$ and $\rangle$ are parenthesis symbols. Thus ordinary finite automata (from now on monoid automata) can be used to define regular binoid languages. More precisely, they defined two kinds of acceptance by monoid automata: the free binoid mode and the free monoid mode. In the case of *free monoid mode* acceptance, given any word $x \in E(\Sigma)^*$ the automaton decides whether x is a valid term representation of a biword in the accepted language. In the case of *free binoid mode* acceptance, the inputs of the automaton just come from the restricted set of valid term representations, and the automaton only decides the question of whether the biword represented by the input term belongs to the accepted language or not. Let $\mathsf{Reg}^{\mathsf{FM}}$ (resp. $\mathsf{Reg}^{\mathsf{FB}}$) denote the class of binoid languages that can be accepted in the free monoid (resp. free binoid) mode.

In [HIJ00] it was shown that $\mathsf{Reg}^{\mathsf{FM}} \subsetneq \mathsf{Reg}^{\mathsf{FB}}$. The main result of [HWJ03] can be expressed in our terminology as $\mathsf{BRat} \subseteq \mathsf{Reg}^{\mathsf{FM}}$. Similarly, [HSJ04] contains the proof of the converse inclusion, namely that $\mathsf{Reg}^{\mathsf{FM}} \subseteq \mathsf{BRat}$. In [HIJ00] phrase structure grammars (B-grammars) generating term representations of binoid languages are also introduced. In particular, they define left and right linear B-grammars and show that these determine different language classes that lie somewhere between finite automata in the free monoid, and the free binoid mode. The paper [HKJ02] deals with finite codes over free binoids.

An obvious advantage of this approach as against parenthesizing automata is that one is not forced to use ordinary automata to describe regular binoid languages. Rather, any equivalent characterization of regular word languages (e.g. regular expressions or MSO-formulas) can be employed instead.

However it should be mentioned that this approach is quite syntactic – which almost always involves long definitions and proofs with many cases. In Section 3.12 we will make a detailed comparison between Hashiguchi's concepts of regularity and ours. In addition, we will propose the use of *condensed terms* (cterms) instead of terms as it can help simplify the concepts and proofs that are monoid-based.

1.4.5 Picture Languages

A different two-dimensional generalization of the classical framework is provided by *picture languages*. Pictures themselves can be viewed as labeled biposets with a very regular rectangular structure. They come with two operations, corresponding to the horizontal and vertical products, but these are only partially defined (cf. [GR96, GR97]). The notion of recognizability is based on tilings and it has different properties, since recognizable picture languages are not closed under complementation and their emptiness

problem is undecidable. For the description of picture languages using formal logic we refer to [GRST94, Wil97].

In [Dol05], Dolinka demonstrated that picture languages and binoid languages satisfy the same identities (for the operations of union, the two products, the two (Kleene) iterations of the two products and some constants). See [Dol07] too for more details about the axiomatization of the equational theory of binoid languages.

1.4.6 Visibly Pushdown and Nested Word Languages

Binoid languages are also closely related to visibly pushdown and nested word languages [AM04, AM06].

The operation of our PA is very similar to that of a visibly pushdown automaton (or VPA for short) [AM04] and a nested word automaton[2] [AM06]. One could imagine that a PA uses a pushdown storage which works in the following way. During an opening parenthesizing transition labeled by $\langle_i$ the automaton puts the index i to the top of the stack, and later the automaton can perform a closing parenthesizing transition labeled $\rangle_i$, only if the index i can be popped from the stack. Notice that a PA alters the stack only when it performs parenthesizing transitions: opening parentheses corresponds to push operations, while closing parentheses corresponds to pop operations. This is what is called visibly pushdown behavior: each input symbol determines independently of the actual state the type of stack operation to perform. This type can be a push or a pop type operation or the machine might leave the stack unchanged. Similarly, each VPA runs on words over a so-called *pushdown alphabet*. This is a triple $\Sigma = \langle \Sigma_c, \Sigma_r, \Sigma_\ell \rangle$, where Σ_c is the *call alphabet*, Σ_r is the *return alphabet*, and Σ_ℓ is the *local alphabet*. A VPA perform a push, pop, or no stack operation by reading letters from Σ_c, Σ_r and Σ_ℓ, in turn. Apart from this a VPA behaves just like an ordinary pushdown automaton. Acceptance is defined by final states.

Thus, it is not hard to see that each PA over $\Sigma^*(\bullet, \circ)$ with i pairs of parentheses naturally corresponds to a VPA with i stack symbols (operating on words over a special pushdown alphabet $\widehat{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_\ell \rangle$, where $\Sigma_c = \{\langle\}$, $\Sigma_r = \{\rangle\}$ and $\Sigma_\ell = \Sigma$, cf. [AM04].) However a PA operates on biwords, not on words. Hence the syntactic check of the input – not just to check the well-balanced aspect of the parentheses, but also to make sure that no empty or superfluous parenthesization occurs – falls outside the task of a PA. Consequently i stack symbols are not enough for a VPA to simulate the behavior of a PA with i pairs of parenthesis, but it can be proved that the set of valid

[2] Actually, the concept of a nested word automaton is a reformulation of that of a VPA.

term or cterm representations can be accepted by a VPA with 3 stack symbols, and hence $3i$ is an upper bound on the number of stack symbols that is necessary.

1.5 The Structure of the Thesis

The rest of the thesis is organized as follows. Chapter 2 introduces biwords and binoid languages, which are the basic objects of our investigations. Here we present both the free algebra theorem and the graph-theoretic characterization within the more general framework of (m, n)-structures of Ésik. Moreover, the representations of biwords, – which are crucial to building the concepts of the next chapters – are also introduced and discussed.

Chapter 3 is devoted to the finitary case, i.e. to the theory of languages of finite biwords. Here we introduce our model of parenthesizing automata which operates on biwords. Then we prove that the expressive power of parenthesizing automata coincides with that of recognizability and MSO-definability. We will investigate various classes of rational binoid languages, and study their relationships.

In Chapter 4 we extend our investigations to infinite biwords. The main result of this chapter is the generalization of the equivalences from the finite case. Namely we prove that with an appropriate generalization of the concept of parenthesizing automata – called parenthesizing Büchi-automata – the equivalence of the recognizable, regular and MSO-definable language classes holds for languages of infinite biwords as well.

Much of the material of this thesis is based on the following publications:

[ÉN04] Z. Ésik and Z. L. Németh, Higher dimensional automata. *J. of Autom. Lang. Comb.* **9** (2004), 3–29.

[ÉN05] Z. Ésik and Z. L. Németh, Algebraic and graph-theoretic properties of infinite n-posets. *Theoret. Informatics Appl.* **39** (2005), 305–322.

[Ném04] Z. L. Németh, A hierarchy theorem for regular languages over free bisemigroups. *Acta Cybern.* **16** (2004), 567–577.

[Ném06] Z. L. Németh, Automata on infinite biposets. *Acta Cybern.* **18** (2006), 765–797.

[Ném07] Z. L. Németh, On the regularity of binoid languages: a comparative approach. In: preproc. *1st Int. Conf. on Language and Automata Theory and Appl., LATA'07*, March 29 – April 4, 2007, Tarragona, Spain.

Chapter 2 contains several ideas taken from the introductory parts of three papers [ÉN04, Ném06, Ném07]. The primary source of Chapter 3 is [ÉN04], but two sections of it, namely Section 3.8 and Section 3.12 present the results given in [Ném04] and [Ném07], respectively. Finally, Chapter 4 is based on the concepts and results given in [ÉN05] and [Ném06].

However the thesis seeks to provide more than just the enumeration of the results of the above papers. It attempts to offer a precise account of the subject of regular binoid languages, with more detailed proofs and examples, along with justifications of the new concepts and conclusions. It also offers a new outlook on solved and unsolved problems, and suggests possible future directions of research.

Chapter 2

Binoids and Biwords

2.1 Preliminaries

In the following, n and m always denote nonnegative integers. We write $[n]$ for the set $\{1, 2, \ldots, n\}$, so $[0]$ means the empty set, also written as $\emptyset$. Moreover, Σ denotes a finite nonempty set, called an *alphabet*, whose elements are called letters, while Σ_m stands for an alphabet that has m letters. The *empty word*, i.e. the word that has no letters, will be denoted by ε. We write Σ^* for the set of all words over Σ, and Σ^+ for the set of all nonempty words over Σ. As usual, $|x|$, the *length* of a word x, is the number of letters in x, and $|x|_\sigma$ stands for the *number of occurrences* of symbol σ in x. We use the notation Σ^n for the set of words over Σ of length n.

The set Ω denotes some finite *set of parentheses*. Of course, Ω and Σ are always disjoint, and elements of Ω are usually written as $\langle_1, \rangle_1, \langle_2, \rangle_2, \ldots$ We also assume here that each Ω is partitioned into sets of *opening and closing parentheses*, denoted by Ω_{op} and Ω_{cl} respectively, which are in bijective correspondence. For any integer $j \geq 0$, let Ω_j stand for a set of j pairs of parentheses, that is $\Omega_j = \{ \langle_1, \rangle_1, \ldots, \langle_j, \rangle_j \}$. Here it is convenient to choose $\Omega_0 := \emptyset$.

Next, let us recall some standard concepts of algebra. A *binary relation* on a set S is a subset $\rho \subseteq S \times S$. If $(x, y) \in \rho$, we write $x \rho y$. We say that a relation ρ on a set S is

- *reflexive* if for all x in S, $x \rho x$ holds;
- *irreflexive* if for all x in S, $x \rho x$ does not hold;
- *symmetric* if for all $x, y \in S$, $x \rho y$ implies $y \rho x$;
- *transitive* if for all $x, y, z \in S$, $x \rho y$ and $y \rho z$ implies $x \rho z$.

A relation ρ that is reflexive, symmetric and transitive is known as an *equivalence relation*. A *(strict) partial order* is a relation that is both irreflexive and transitive. A

partial order ρ on a set S such that for all elements $x, y \in S$ either $x \rho y$ or $y \rho x$ holds, is called a *linear order*.

As usual a *graph* or *undirected graph* is a pair $G = (V, E)$, where V is a set of *vertices* and E is a set of unordered pairs of distinct vertices, called *edges*. If E is a set of ordered pairs of vertices, we have a *directed graph* or *digraph*. A *walk* in a (directed or undirected) graph is a sequence of vertices such that from each vertex in the sequence there is an edge to the next vertex in the sequence. A walk with no repeated vertices is called a *path*. A walk in which only the first and the last vertices are the same is referred to as a *cycle*. Graphs without cycles are called *acyclic* graphs. *DAG* is an abbreviation for directed acyclic graphs.

An n-ary *operation* on a set A is a function $A^n \to A$, $(n \geq 0)$. An *algebraic structure*, or *algebra* for short, is a set A together with a collection of operations on A. The 2-ary operations are also called *binary operations*, and often denoted by infix notation like $x \circ y$. A binary operation $\circ$ on A is commutative if $a \circ b = b \circ a$, for all a and b in A. Moreover, $\circ$ is associative if $(a \circ b) \circ c = a \circ (b \circ c)$, for all a, b and c in A. A *neutral element* or *identity* of a binary operation $\circ$ on A is an element e in A such that $e \circ a = a \circ e = a$ holds for all a in A. If A is an algebra and ρ is an equivalence relation on A, then ρ is called a *congruence* of $\mathcal{A}$ if it is invariant under all operations of $\mathcal{A}$. E.g. if $\circ$ is a binary operation, then $x_1 \,\rho\, x_2$ and $y_1 \,\rho\, y_2$ imply $(x_1 \circ y_1) \,\rho\, (x_2 \circ y_2)$.

2.2 Bisemigroups and n-semigroups

Definition 2.1 *We call a set equipped with n associative binary operations an* n-semigroup. *The usual notation for an n-semigroup is $(S, \circ_1, \ldots, \circ_n)$. A* bisemigroup *is an n-semigroup for $n = 2$. The two binary operations of a bisemigroup are called the* horizontal product *and the* vertical product. *They are denoted by $\bullet$ and $\circ$, respectively. A bisemigroup is usually written as $(B, \bullet, \circ)$.*

It is well-known [BS81] that for every alphabet Σ there is an (up to isomorphism unique) bisemigroup that is freely generated by Σ in the variety of all bisemigroups. This bisemigroup is called the *free bisemigroup* (generated by Σ), and is written as $\Sigma^+(\bullet, \circ)$.

Now we give some examples for bisemigroups. We have already seen an example in the Introduction. In fact, nonempty two-dimensional words represent the elements of the free bisemigroups. We will discuss free bisemigroups in more detail later on in Section 2.6. Obviously, more general algebraic structures like semirings, rings and fields

can also be regarded as bisemigroups. One algebraic drawback of picture languages [GR97] is that for pictures the concatenation operations are only partially defined since, for example, we cannot form the horizontal product of two pictures if they have a different number of rows. But if we add a common zero element for both operations, and we require the result of all formerly undefined products to be this new zero element, then we get a bisemigroup as well. Note that this bisemigroup is not free.

The next example is the modeling of concurrent processes similar to that achieved by sp-posets, but in the case when parallel composition is just associative, but not commutative. This may happen if, when we form a parallel composition, we always give an order of priority to the operands. In addition, we demand that if there is insufficient free processor capacity to start both operand subprocess at the same time, then the subprocess with the higher priority shall be started first. Naturally, if sufficient free capacity is present at the beginning or is released later, then the two subprocesses can run in parallel.

2.3 Binoids and n-monoids

Definition 2.2 *An algebraic structure* $(S, \circ_1, \ldots, \circ_n, 1)$ *is called an* n-monoid*, if* $(S, \circ_1, \ldots, \circ_n)$ *is an* n*-semigroup and* $1 \in S$ *is an identity element for all operations, i.e.* $1 \circ_i s = s \circ_i 1 = s$*, for all* $s \in S$*,* $i \in [n]$*. A* binoid *is an* n*-monoid for* $n = 2$*. Hence a binoid is usually written as* $(B, \bullet, \circ, 1)$*.*

As before, for every alphabet Σ there is a (unique up to an isomorphism) binoid that is freely generated by Σ in the variety of all binoids. This binoid is called the *free binoid* (generated by Σ) and is written as $\Sigma^*(\bullet, \circ)$.

Remark 2.3 Hashiguchi et al. have also defined algebras with two associative operations and two identities, where the two identities are not necessarily the same. In [HIJ00, HWJ03, HSJ04] these algebras are called *bimonoids* and of course they are not the same as our 2-monoids. But our 2-monoids (or binoids) are also called binoids in [HIJ00, HWJ03, HSJ04]. On the other hand, 2-monoids are called *double monoids* in [Gra81].

By definition every binoid can be regarded as a bisemigroup. Also, every bisemigroup can be transformed to a binoid by adding a new identity element that serves as an identity for both operations. Thus all of our previous examples of bisemigroups are examples of binoids if we add an identity element to them.

Automata in general operate on elements of some free algebra. In the classical case, automata run on words, i.e. on elements of free semigroups (or free monoids if we also

consider the empty word.) Hence if we wish to generalize automata to higher dimensions, it is natural to examine how they can operate on elements of free n-semigroups (or free n-monoids). Thus in the rest of this chapter we will study free n-semigroups and n-monoids, and representations of their elements. However, a fundamental observation is that it is enough to just consider the case $n = 2$, as almost all of our results can be extended to the general case in a straightforward way. Unfortunately, there are some exceptions where the generalization is not so evident, but the reader should keep this in mind.

Thus, free binoids and bisemigroups will be the central objects of our investigations. There is no substantial difference between them, since adding or removing the identity will have no influence on whether a given language belongs to the classes under study. Earlier Hashiguchi et al. have developed their theory on free binoids, while in the previous work of Ésik and Németh free bisemigroups were used. In order to facilitate a comparison between these two approaches, in the thesis, except for Chapter 4 (cf. Remark 4.1), we will employ free binoids.

2.4 (m,n)-structures

Before discussing free binoids and their elements, we will consider the more general framework of (m, n)-semigroups and (m, n)-structures introduced by Ésik. This general setting allows one to consider any finite number of operations that are associative, or associative and commutative. The following definitions and results are from [Ési00]. In the rest of this section we will assume that $m + n \geq 1$.

Definition 2.4 (cf. Definition 2.5 of [Ési00]) *An* (m, n)-semigroup *is an algebra* $(S, \odot_1, \ldots, \odot_m, \otimes_1, \ldots, \otimes_n)$*, where* $\odot_1, \ldots, \odot_m$ *are associative, and* $\otimes_1, \ldots, \otimes_n$ *are associative and commutative binary operations on* S*.*

Definition 2.5 (Definition 2.1 of [Ési00]) *An* (m, n)-structure *over* Σ*, or just* (m, n)*-structure, for short, is a finite set* P *of vertices equipped with* m *irreflexive transitive relations* $<_1, \ldots, <_m$*,* n *irreflexive symmetric relations* $\sim_1, \ldots, \sim_n$*, and a labeling function* $\lambda : P \to \Sigma$ *subject to the following condition: for any two distinct vertices* $x, y \in P$*, either there is a unique* $i \in [m]$ *with* $x <_i y$ *or* $y <_i x$*, or else there is a unique* $j \in [n]$ *with* $x \sim_j y$*. A* homomorphism *of* (m, n)*-structures is a function which preserves the relations and the labeling. An* isomorphism *is a bijective homomorphism.*

Remark 2.6 Note that (m, n)-structures are indeed quite general objects. It can easily be verified that, by definition, there is a bijective correspondence between

- $(1,0)$-structures and labeled linear orders (words),
- $(0,1)$-structures and multisets,
- $(0,2)$-structures and labeled graphs,
- $(1,1)$-structures and labeled partial orders (posets),
- $(n,1)$-structures and labeled n-posets[1], where $n \geq 1$.

As usual, we do not distinguish between isomorphic (m,n)-structures, so that in the definition below we may assume that P and Q are disjoint.

Definition 2.7 (Definition 2.3 of [Ési00]) *Suppose that* $P = (P, <_1^P, \ldots, <_m^P, \sim_1^P, \ldots, \sim_n^P, \lambda^P)$ *and* $Q = (Q, <_1^Q, \ldots, <_m^Q, \sim_1^Q, \ldots, \sim_n^Q, \lambda^Q)$ *are disjoint* (m,n)*-structures. For each* $i \in [m]$*, we define* $P \odot_i Q$ *to be the* (m,n)*-structure* $(P \cup Q, <_1, \ldots, <_m, \sim_1, \ldots, \sim_n, \lambda)$ *with*

$$
\begin{aligned}
<_k &= <_k^P \cup <_k^Q, \quad k \neq i, k \in [m] && (2.1)\\
<_i &= <_i^P \cup <_i^Q \cup (P \times Q) && (2.2)\\
\sim_j &= \sim_j^P \cup \sim_j^Q, \quad j \in [n] \\
\lambda &= \lambda^P \cup \lambda^Q. && (2.3)
\end{aligned}
$$

For each $j \in [m]$*, let* $P \otimes_j Q$ *be the* (m,n)*-structure* $(P \cup Q, <_1, \ldots, <_m, \sim_1, \ldots, \sim_n, \lambda)$ *with*

$$
\begin{aligned}
<_i &= <_i^P \cup <_i^Q, \quad i \in [m] \\
\sim_k &= \sim_k^P \cup \sim_k^Q, \quad k \neq j, k \in [n] \\
\sim_j &= \sim_j^P \cup \sim_j^Q \cup (P \times Q) \cup (Q \times P) \\
\lambda &= \lambda^P \cup \lambda^Q.
\end{aligned}
$$

Example 2.8 Note that each $\odot_i$ is associative operation and each $\otimes_j$ is associative and commutative, but associativity does not hold for different operations, e.g. $a \odot_1 (b \odot_2 c) \neq (a \odot_1 b) \odot_2 c$. That is why writing $a \odot_1 b \otimes_1 c \odot_2 d$ without parentheses is ambiguous. Actually, it is known that $n+1$ factors can be completely parenthesized in C_n number of ways, where $C_n = \frac{1}{n+1}\binom{2n}{n}$ is the n^{th} Catalan number [WikA, Sta99]. Hence from $a \odot_1 b \otimes_1 c \odot_2 d$ we can have $C_3 = 5$ different $(2,1)$-structures using different parenthesizations. These are depicted in Figure 2.1.

In what follows we will identify the singleton (m,n)-structures by their letters and assume that all (m,n)-structures are over a common, fixed alphabet Σ.

[1] n-posets are labeled sets equipped with n partial order relations (see Definition 2.12).

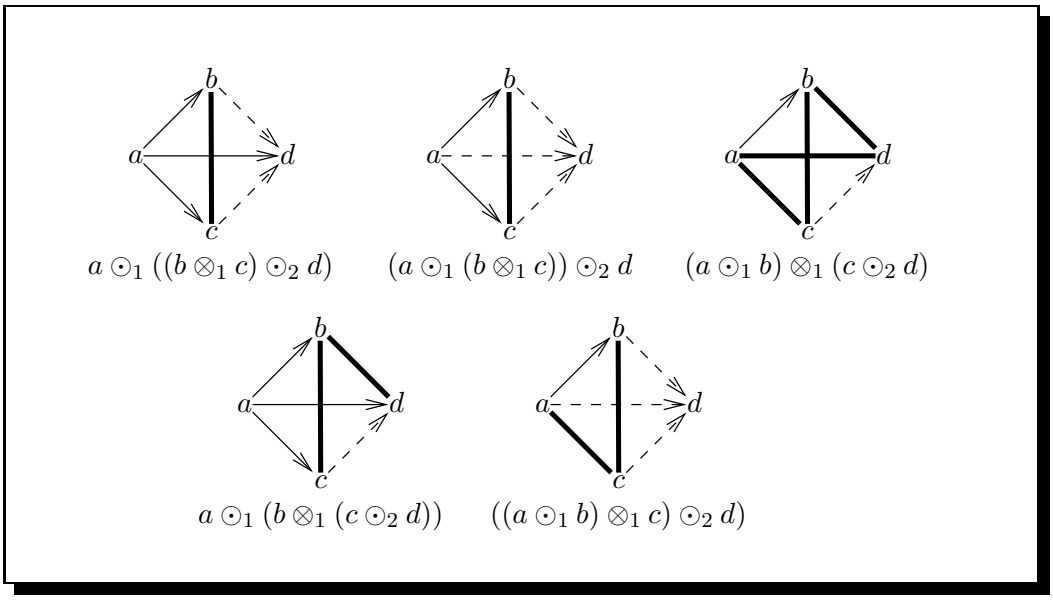

Figure 2.1: *Those* $(2,1)$*-structures that can be obtained from* $a \odot_1 b \otimes_1 c \odot_2 d$ *using different parenthesizations. Relations* $<_1$*,* $<_2$ *and* $\sim_1$ *are represented by solid arrows, dashed arrows and thick lines, respectively.*

Definition 2.9 (Definition 2.6 of [Ési00]) *An* (m,n)*-structure is* reducible *if it can be generated from the singletons corresponding to the letters in* Σ *by the* $m+n$ *product operations, i.e. when it can be reduced to a singleton by decomposition with respect to the operations.*

Let $\mathbf{S}_\Sigma$ denote the set of all (m,n)-structures and let $\mathbf{RS}_\Sigma$ be the set of all reducible (m,n)-structures over Σ. It is clear that with the product operations of Definition 2.7 $\mathbf{S}_\Sigma$ is an (m,n)-semigroup, and $\mathbf{RS}_\Sigma$ is the least subalgebra of those subalgebras of $\mathbf{S}_\Sigma$ that contain Σ.

2.5 Free Algebra Theorems

The notion of a free algebra is one of the fundamental concepts of universal algebra [BS81]. It can be defined at a rather abstract level, – even in terms of category theory – but here we introduce it only for (m,n)-semigroups. We will use a well-known characterization called the universal property.

Definition 2.10 *An* (m,n)*-semigroup* S *is called* freely generated by some set $A \subseteq S$ in the variety of all (m,n)-semigroups*, if for every* (m,n)*-semigroup* S' *and mapping*

$h : A \to S'$, there is a unique extension of h to a homomorphism $h^{\#} : S \to S'$.

Theorem 2.11 (Theorem 2.8 of [Ési00]) $\mathbf{RS}_\Sigma$ *is freely generated by the set Σ in the variety of all (m, n)-semigroups.*

Proof. The proof of the above theorem is based on the following two properties of reducible (m, n)-structures:
Property 1: Each reducible (m, n)-structure in $\mathbf{S}_\Sigma$ can be constructed by the $m + n$ product operations from the singleton structures.
Property 2: For each reducible (m, n)-structure, its construction from the singletons mentioned above (in Property 1) is unique apart from the associativity of the $\odot_i$ and the associativity and commutativity of the $\otimes_j$ operations ($i \in [m]$, $j \in [n]$).

Property 1 is an immediately consequence of the definition of reducibility. Property 2 follows from the fact that the operations are "invertible" in the sense that a product $P \odot_i Q$ ($i \in [n]$) cannot be decomposed into any other product $P' \odot_j Q'$ or $P'' \otimes_k Q''$, for any $i \neq j \in [n]$ or $k \in [m]$. Of course it may happen that $P \odot_i Q = P' \odot_i Q'$ holds with $P \neq P'$ and $Q \neq Q'$. But then both products must have a common refinement $P \odot_i Q = P' \odot_i Q' = P_1 \odot_i P_2 \odot_i \ldots \odot_i P_k$, so further decompositions of the operands will lead to the same result. These two properties make it possible for us to prove the universal property of $\mathbf{S}_\Sigma$. Indeed, for any (m, n)-structure S', and any mapping $h : \Sigma \to S'$, by Property 1, $h^{\#}(P)$ can be defined according to the decomposition of P from the singletons. To compute $h^{\#}(P)$ one should apply the same operations in the same order as P is built from the singletons, but using the images $h(\sigma) \in S'$ instead of the singleton $\sigma \in S$. Finally, it is not hard to see that Property 2 ensures that $h^{\#}$ is a homomorphism, as required. □

As was mentioned in Remark 2.6, (m, n)-structures include many important special cases. Hence Theorem 2.11 can also be used to give a description of the elements of the free algebras in these special situations. These are summarized in Figure 2.2. First, $(1, 0)$-semigroups and $(1, 0)$-structures correspond to the classical theory of semigroups and (nonempty) words. Next, $(n, 0)$-semigroups (i.e. when we have n associative operations, and no operation that is both associative and commutative) are the *n-semigroups* of Definition 2.1. Free n-semigroups can be described with the help of n-posets defined as follows.

Definition 2.12 *For any integer $n \geq 1$, an* n-poset *over Σ is a structure $(P, <_1, \ldots, <_n, \lambda)$, where P is a set, $<_1, \ldots, <_n$, are strict partial order relations on P, and λ is a labeling function $\lambda : P \to \Sigma$.*

$*$	**Algebraic structures:**	**Elements of the free algebras:**
$(1,0)$	semigroups	nonempty words
	$\downarrow$	$\downarrow$
$(2,0)$	bisemigroups	sp-biposets
	$\downarrow$	$\downarrow$
$(n,0)$	n-semigroups	constructible n-posets
	$\downarrow$	$\downarrow$
(m,n)	(m,n)-semigroups [Ési00]	reducible (m,n)-structures
	$\uparrow$	$\uparrow$
$(1,1)$	sp-algebras [LW98]	sp-posets

$*$ The number of associative, and associative & commutative operations.

Figure 2.2: *Some algebras and their elements. Here arrows mean generalizations.*

On n-posets n product operations can be defined exactly as in (2.1), (2.2) and (2.3). Note that, in contrast to our definition of (m,n)-structures, here we do not require that any two elements are related by exactly one relation. Still, it can be easily shown that *constructible n-posets*, (i.e. those that can be constructed from the singletons, and hence may be identified with reducible $(n,0)$-structures) have the above-mentioned totality property.

As we said earlier, the primary aim of the thesis is to develop an automata theory over free n-monoids for any $n \geq 2$, but it is usually enough to consider just the $n = 2$ case. Accordingly, we will focus on constructible 2-posets. The term *biposet* is a synonym for 2-poset, and constructible 2-posets will also be called *series-parallel biposets* or simply *sp-biposets*.

Next, another important special case of (m,n)-semigroups and (m,n)-structures is the situation when $m = n = 1$. Indeed, it is straightforward to identify a $(1,1)$-structure $(P, <_1, \sim_1, \lambda)$ with the labeled poset $(P, <_1, \lambda)$. Posets that correspond to the reducible $(1,1)$-structures are called *series-parallel posets*, or *sp-posets* for short, and in [LW98, LW00, LW01] $(1,1)$-semigroups are called *sp-algebras*.

2.6 Representations of Biwords

2.6.1 Preliminaries

First let us recall that $\Sigma^*(\bullet, \circ)$ denotes the free binoid generated by Σ, and the two operations of $\Sigma^*(\bullet, \circ)$ are called the horizontal product and the vertical product. The

elements of $\Sigma^*(\bullet, \circ)$ are called *biwords* (over Σ). In what follows we will use the term biword in an abstract sense, but we will give concrete representations of it later on. The identity of $\Sigma^*(\bullet, \circ)$, denoted by ε, is the *empty biword*. Each generator of $\Sigma^*(\bullet, \circ)$ corresponding to a letter $\sigma \in \Sigma$ is called a *singleton biword* and will also be denoted by σ. The biwords that can be written as a horizontal (or vertical) product of two nonempty biwords are called *horizontal* (or *vertical*). We call this property the *type* of a biword. If a biword w is written in the form $w = w_1 \bullet w_2 \bullet \ldots \bullet w_n$, where each w_i is nonempty, then this form is called a *horizontal decomposition* of w. Moreover, if each w_i is either a singleton or a vertical biword, then the decomposition is called *maximal*. Obviously every horizontal biword has a unique maximal horizontal decomposition. For vertical biwords *vertical decompositions* and *maximal vertical decompositions* are defined symmetrically.

Of course there are several possible ways of describing biwords. They may be drawn on a paper as two-dimensional words using boxes as we did in the Introduction. In the previous section the sp-biposet representation was introduced. Biwords may also be regarded as labeled ordered unranked trees. Furthermore, we will also employ two linear representations, namely terms and condensed terms.

A common feature of all the representations is that the elements are constructed inductively from the representations of the one-letter biwords by the two products operations. Moreover, these constructions are unique up to the associativity of the operations. In other words, the analogous two properties in the proof of Theorem 2.11 hold. Hence in every case we get the same algebra: $\Sigma^*(\bullet, \circ)$, the free algebra of biwords over Σ.

Next, we will also consider the description of biwords by i-terms and i-cterms, which are ambiguous in the sense that a biword may have more than one i-(c)term representation.

2.6.2 The Sp-biposet Representation

First of all, series-parallel biposets, or sp-biposets for short, are just constructible $(2, 0)$-structures. Therefore, from Theorem 2.11 we immediately see that they represent the elements of the free bisemigroups. Moreover, if we include the empty biposet as well, they will represent the elements of the free binoids.

For reasons which will become clearer later on, we will reformulate their definition starting from the notion of biposets instead of $(2, 0)$-structures. This will help clarify the reason for the name "series-parallel" and we also fix both notation and terminology. If we choose $n = 2$ in Definition 2.12, we get the notion of biposet.

Definition 2.13 *A* Σ-labeled biposet*, or* biposet *for short, is a 4-tuple* $(P, <_h^P, <_v^P, \lambda_P)$*, where* P *is a finite set of vertices,* $<_h^P$ *and* $<_v^P$ *are strict partial orders on* P *and* $\lambda_P : P \to \Sigma$ *is a labeling function. If* $P = \emptyset$*, we get the empty biposet, denoted by* ε*.*

We say that two biposets are *isomorphic* if there is a bijective function on the vertices that preserves the partial orders and the labeling. Below we will make no distinctions between isomorphic biposets.

Suppose that $P = (P, <_h^P, <_v^P, \lambda_P)$ and $Q = (Q, <_h^Q, <_v^Q, \lambda_Q)$ are Σ-labeled biposets. Without loss of generality, assume that P and Q are disjoint. We define their *horizontal or series product* as $P \bullet Q = (P \cup Q, <_h^{P \bullet Q}, <_v^{P \bullet Q}, \lambda_{P \bullet Q})$, and their *vertical or parallel product* as $P \circ Q = (P \cup Q, <_h^{P \circ Q}, <_v^{P \circ Q}, \lambda_{P \circ Q})$, where

$$\begin{array}{lcllcl} <_h^{P \bullet Q} & = & <_h^P \cup <_h^Q \cup (P \times Q), & <_h^{P \circ Q} & = & <_h^P \cup <_h^Q, \\ <_v^{P \bullet Q} & = & <_v^P \cup <_v^Q, & <_v^{P \circ Q} & = & <_v^P \cup <_v^Q \cup (P \times Q), \end{array}$$

and the labeling are $\lambda_{P \bullet Q} = \lambda_{P \circ Q} = \lambda_P \cup \lambda_Q$.

It is quite apparent that both product operations are associative. Each letter $\sigma \in \Sigma$ may be identified by the singleton biposet labeled σ. Now, let $\mathrm{SPB}(\Sigma)$ denote the collection of biposets that can be generated from the singletons corresponding to the letters in Σ by the two product operations. The biposets in $\mathrm{SPB}(\Sigma)$ are called *series-parallel biposets*, or *sp-biposets* for short.

The name "series-parallel" comes from the theory of posets. From a practical point of view series-parallel posets can be viewed as models of modularly constructed concurrent processes (cf. [Pra86, Nie01, LW98, LW00]). In this modeling the two product operations are the series product (which refers to the subsequent execution of two subprocesses) and the parallel product (which models concurrent execution). In an analogous way, our two products on biposets are also called the series product and parallel product. However, unlike the parallel composition of posets, our parallel product is not commutative. So perhaps the terms "horizontal and vertical" are more satisfactory, as they make no distinction between the two operations. Nevertheless, for the generated objects we will retain the name "series-parallel biposets" as they have a close connection with series-parallel posets and series-parallel graphs.

Now we will make use of the free algebra theorem (Theorem 2.11) for the special case of bisemigroups and sp-biposets.

Theorem 2.14 (cf. proposition 2.3 of [ÉN02]) *The bisemigroup* $\mathrm{SPB}(\Sigma)$ *is freely generated by* Σ *in the variety of all bisemigroups. Thus* $(\mathrm{SPB}(\Sigma), \bullet, \circ)$ *is isomorphic to* $\Sigma^+(\bullet, \circ)$*, and* $(\mathrm{SPB}(\Sigma) \cup \{\varepsilon\}, \bullet, \circ)$ *is isomorphic to* $\Sigma^*(\bullet, \circ)$*.*

The *sp-biposet representation* of a biword w will be denoted by w^{bp}. An example of the sp-biposet representation of a biword will be given in Example 2.18.

2.6.3 The Term Representation

Perhaps the most obvious way of representing biwords is by the use of terms. Here we describe how to associate a term w^{tm} with each biword $w \in \Sigma^*(\bullet, \circ)$. To this end, we will extend the alphabet Σ with operation symbols and parentheses. Let $E(\Sigma) := \Sigma \cup \{ \bullet, \circ, \langle, \rangle \}$. As usual, in the term representation we shall put parentheses around the subterm of horizontal biwords which appear as a vertical factor, and symmetrically around the subterm of vertical biwords which appear as a horizontal factor. This procedure can be stated more precisely in the following way.

Definition 2.15 *If $w \in \Sigma^*(\bullet, \circ)$, then w^{tm} will denote the* term representation *of w. Let w^{tm} be a word over $E(\Sigma)$, defined inductively as follows.*

(i) *If $w = \varepsilon$ is the empty biword, then $w^{tm} := \varepsilon$.*

(ii) *If $w = \sigma$ is a singleton biword, then $w^{tm} := \sigma$.*

(iii) *If $w = w_1 \bullet w_2$ with $w_1, w_2 \neq \varepsilon$, then $w^{tm} := \mathrm{Hform}(w_1) \bullet \mathrm{Hform}(w_2)$.*

(iv) *If $w = w_1 \circ w_2$ with $w_1, w_2 \neq \varepsilon$, then $w^{tm} := \mathrm{Vform}(w_1) \circ \mathrm{Vform}(w_2)$.*

In (iii), $\mathrm{Hform}(w)$ *denotes the* horizontal form *of the biword w, defined as:*

$$\mathrm{Hform}(w) \quad := \quad \begin{cases} w^{tm} & \text{if } w \text{ is a singleton or horizontal biword,} \\ \langle\, w^{tm} \,\rangle & \text{if } w \text{ is a vertical biword.} \end{cases}$$

In (iv), $\mathrm{Vform}(w)$, *the* vertical form *of w, is defined symmetrically.*

It should be mentioned here that in cases (iii) and (iv) the definition of w^{tm} does not depend on the choice of factorization since each $\bullet$, $\circ$ and the concatenation of words are associative operations.

2.6.4 The Condensed Term Representation

Another description of $\Sigma^*(\bullet, \circ)$ can be given by using *condensed terms*, or *cterms* for short. This representation may not be interesting by itself. However, its importance comes from the fact that our automata operate on condensed terms – processing them sequentially – reading from left to right.

The condensation of the description of the terms is based on a simple observation. It is that the operation symbols can be omitted provided we know the type of a biword in advance. Actually, the arrangement of the parentheses tell us precisely where we should put the horizontal and vertical product operations between the factors. Formally, condensed term representations are words from the set

$$\{\,\varepsilon\,\} \cup \Sigma \cup \{\,\bullet, \circ\,\}\Big(\Sigma \cup \{\,\langle, \rangle\,\}\Big)^{+}.$$

The representation of the empty biword and the singletons are obvious. For a nonempty and nonsingleton biword, the first letter – the *type-sign* – gives the type of the represented biword; namely $\bullet$ and $\circ$ mean the horizontal type and vertical type, respectively. The remaining part is just the term representation after the operation symbols have been deleted. Here we shall write w^{ctm} for the cterm representation of biword w. Thus if $w^{tm} = \langle a \bullet b\rangle \circ \langle c \bullet \langle d \circ e \circ f\rangle \bullet g\rangle$, then $w^{ctm} =\circ \langle ab\rangle\langle c\langle def\rangle g\rangle$.

Hence it is possible to recover the original term representation from a condensed one. To achieve this, all we need to do is put back the operation symbols that were omitted. On the outer level the type of the operations is given explicitly by the first letter. Moreover, moving from left to right the type of the operation symbols alternates between the parentheses. Finally, we should put an operation symbol after each letter and closing parenthesis symbols, provided they are followed by another letter or by an opening parenthesis symbol.

Sometimes it will be useful to write w^{ctm-} for the word we get from w^{ctm} by deleting the type-sign of w. In the case of the empty biword and singletons there are no type-signs, so let $w^{ctm-} = w^{ctm}$ in such cases.

We can extend these notations to languages as well. Let $L^{tm} := \{\, w^{tm} \mid w \in L\,\}$ and $L^{ctm} := \{\, w^{ctm} \mid w \in L\,\}$. After, let $\mathrm{TM}(\Sigma) := \Sigma^*(\bullet, \circ)^{tm}$ and $\mathrm{CTM}(\Sigma) := \Sigma^*(\bullet, \circ)^{ctm}$ denote the set of all (c)terms of biwords over Σ.

Example 2.16 The term representation of the biword depicted in Figure 1.1 is

$$w^{tm} = a \bullet \langle b \circ \langle c \bullet d\rangle\rangle \bullet \langle e \circ f\rangle.$$

Now, the condensed term representation of this biword is as follows:

$$w^{ctm} =\bullet a\langle b\langle cd\rangle\rangle\langle ef\rangle$$

In addition, we have

$$w^{ctm-} = a\langle b\langle cd\rangle\rangle\langle ef\rangle.$$

2.6.5 The Tree Representation

We will also use finite ordered unranked trees to represent biwords. In this case, leaves are labeled by letters from Σ, and the inner nodes are labeled by $\bullet$ or $\circ$. In addition, $\bullet$ and $\circ$ alternate along any path from the root to a leaf.

Definition 2.17 *If w is a biword, its* tree form w^{tr} *is defined as follows.*

(i) *If $w = \varepsilon$, the empty biword, then w^{tr} is the empty tree, also denoted by ε.*

(ii) *If $w = \sigma$ is a singleton, then w^{tr} is a tree consisting of a single vertex labeled by σ.*

(iii) *If w is a horizontal biword, consider the maximal horizontal decomposition $w = w_1 \bullet w_2 \bullet \ldots \bullet w_m$, $(m \geq 2)$. Now w^{tr} is the tree whose root is labeled by $\bullet$ and this root connects the subtrees w_1^{tr}, w_2^{tr}, ..., w_m^{tr} (in this order).*

(iv) *If w is a vertical biword, consider the maximal vertical decomposition $w = w_1 \circ w_2 \circ \ldots \circ w_m$, $(m \geq 2)$. Now w^{tr} is the tree whose root is labeled by $\circ$ and this root connects the subtrees w_1^{tr}, w_2^{tr}, ..., w_m^{tr} (in this order).*

Example 2.18 Recall our previous example of a biword, namely w whose term representation is

$$w^{tm} = a \bullet \langle b \circ \langle c \bullet d \rangle \rangle \bullet \langle e \circ f \rangle.$$

The two-dimensional word, sp-biposet and tree representations of w are depicted in Figure 2.3. The sp-biposet representation of w is $w^{bp} = (\{1, 2, \ldots, 6\}, <_h, <_v, \lambda)$, where $<_h$ and $<_v$ are the transitive closures of the relations $1 <_h 2$, $1 <_h 3$, $3 <_h 4$, $2 <_h 5$, $2 <_h 6$, $4 <_h 5$, $4 <_h 6$, and $2 <_v 3, 2 <_v 4, 5 <_v 6$, respectively. Moreover, $\lambda(1) = a$, $\lambda(2) = b$, $\lambda(3) = c$, $\lambda(4) = d$, $\lambda(5) = e$, $\lambda(6) = f$. In the figure, horizontal and vertical relations are indicated by solid arrows and dashed arrows, respectively.

It is evident that for any leaf node in w^{tr} there is a corresponding vertex in w^{bp}. Hence we may and will identify the leaves of w^{tr} by the corresponding vertices of w^{bp}. This allows us to speak about elements and subsets of w^{bp} like those of w^{tr}. Similarly, we can associate vertices of w^{bp} with the corresponding letters in the (c)term representation w^{tm} and w^{ctm}.

2.6.6 i-term and i-cterm Representations

As we shall see, in order to accept all recognizable binoid languages by either monoid automata or parenthesizing automata, it is necessary to employ several pairs of parentheses. For this reason choose an integer $i \geq 0$, recall that $\Omega_i = \{ \langle_1, \rangle_1, \ldots, \langle_i, \rangle_i \}$, and

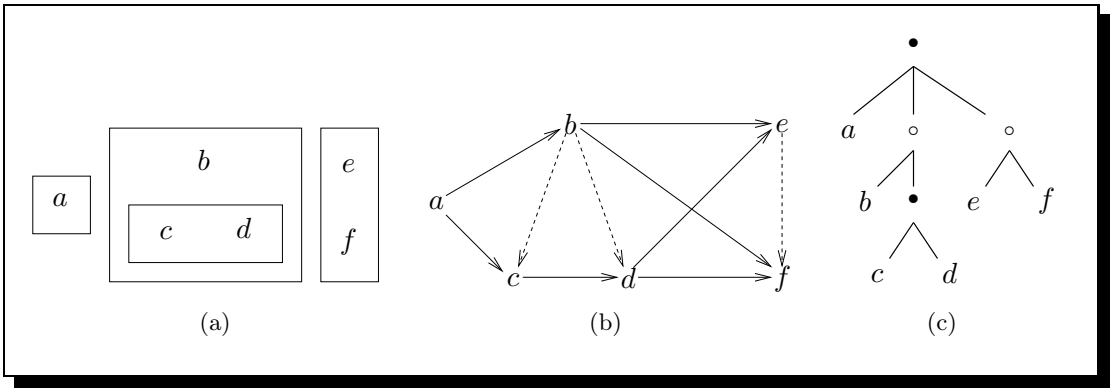

Figure 2.3: *The two-dimensional word representation (a); the biposet representation (b); and the tree representation (c) of the biword* $a \bullet \langle b \circ \langle c \bullet d \rangle \rangle \bullet \langle e \circ f \rangle$.

let $E_i(\Sigma) = \Sigma \cup \Omega_i \cup \{\bullet, \circ\}$ be the *extended alphabet with i different pairs of parentheses*. Next, suppose that w^{tm} (or w^{ctm}) is a (c)term representation of a biword $w \in \Sigma^*(\bullet, \circ)$. Now i-term (or i-cterm) representations of w are obtained by replacing the matching pairs of parentheses with some pairs of indexed parentheses from Ω_i in w^{tm} (resp. in w^{ctm}). Note that a biword can have several different i-(c)term representations. For instance $\langle_2 a \bullet b\rangle_2 \circ \langle_1 c \bullet d\rangle_1$ and $\langle_1 a \bullet b\rangle_1 \circ \langle_1 c \bullet d\rangle_1$ are both 2-term representations of the biword $\langle a \bullet b\rangle \circ \langle c \bullet d\rangle$. Now let $\mathrm{TM}_i(\Sigma)$ and $\mathrm{CTM}_i(\Sigma)$ stand for the i-term and i-cterm representations of the biwords in $\Sigma^*(\bullet, \circ)$, respectively. For a binoid language $L \subseteq \Sigma^*(\bullet, \circ)$, any word language $L' \subseteq \mathrm{TM}_i(\Sigma)$ (resp. $L' \subseteq \mathrm{CTM}_i(\Sigma)$) such that $\eta_i(L') = L^{tm}$ (resp. $\eta_i(L') = L^{ctm}$) shall be referred to as an *i-term representation* (resp. *i-cterm representation*) of L. Here η_i is the mapping that deletes the indices of the parentheses, i.e. the homomorphism $\eta_i : E_i(\Sigma)^* \to E(\Sigma)^*$ which extends

$$\widetilde{\eta}_i(x) = \begin{cases} x & \text{if } x \in \Sigma \cup \{\bullet, \circ\}; \\ \langle & \text{if } x \in \Omega_{i,op}; \\ \rangle & \text{if } x \in \Omega_{i,cl}, \end{cases} \qquad \text{for all } x \in E(\Sigma).$$

2.6.7 Characterizations of $\mathrm{TM}_i(\Sigma)$ and $\mathrm{CTM}_i(\Sigma)$

Before stating the theorems that characterize $\mathrm{TM}_i(\Sigma)$ and $\mathrm{CTM}_i(\Sigma)$ we need to introduce some technical definitions. First, we will define a homomorphism which deletes all the letters of Σ and the operation symbols from the words over $E_i(\Sigma)$. To this end, let $\pi_i : E_i(\Sigma)^* \to \Omega_i^*$ be the unique extension of the mapping

$$\pi_i(x) = \begin{cases} x & \text{if } x \in \Omega_i, \\ \varepsilon & \text{if } x \in \Sigma \cup \{\bullet, \circ\}. \end{cases}$$

For any $i > 0$, the *Dyck language* (cf. [Har78]) over the alphabet Ω_i, denoted Dyck_i, is defined by the context free grammar with start symbol S and rules

$$S \to SS \mid \langle_1 S\rangle_1 \mid \langle_2 S\rangle_2 \mid \ldots \mid \langle_i S\rangle_i \mid \varepsilon.$$

It is well-known and is not hard to see that Dyck_i consists of all the balanced strings of parentheses over Ω_i. Now suppose that $x \in E_i(\Sigma)^*$. The number of opening and closing parentheses in x is then

$$|x|_\langle = \sum_{1\le k\le i} |x|_{\langle_k}, \quad \text{and} \quad |x|_\rangle = \sum_{1\le k\le i} |x|_{\rangle_k}.$$

Furthermore, for all $1 \le j \le |x|$, we let $\mathrm{dp}_x(j) := |x_1 \ldots x_j|_\langle - |x_1 \ldots x_j|_\rangle$. Thus $\mathrm{dp}_x(j)$ can be regarded as the depth of the parenthesization at the jth position in x. Now we will characterize the elements of $\mathrm{TM}_i(\Sigma)$ and $\mathrm{CTM}_i(\Sigma)$ in $E_i(\Sigma)^+$.

Lemma 2.19 *For all $x \in E_i(\Sigma)^+$ we have $x \in \mathrm{TM}_i(\Sigma)$ iff the following conditions hold.*

(1) *$\pi_i(x) \in \mathrm{Dyck}_i$, i.e. the use of parentheses in x is balanced.*

(2) *$x \neq x_1\langle_k\langle_l x_2\rangle_l\rangle_k x_3$, for any $\langle_k, \rangle_k, \langle_l, \rangle_l \in \Omega_i$, and $x_1, x_2, x_3 \in E_i(\Sigma)^*$ such that $\pi_i(x_2) \in \mathrm{Dyck}_i$, i.e. there is no* double parenthesization *in x.*

(3) *$x \neq \langle_k x'\rangle_k$, for any $\langle_k, \rangle_k \in \Omega_i$ and $x' \in E_i(\Sigma)^*$ such that $\pi_i(x') \in \mathrm{Dyck}_i$, i.e. there is no* outer parenthesization *in x.*

(4) *$x \neq x_1\langle_k\sigma\rangle_k x_2$, for any $\langle_k, \rangle_k \in \Omega_i$, $x_1, x_2 \in E_i(\Sigma)^*$ and $\sigma \in \Sigma$, i.e. there is no* singleton parenthesization *in x.*

(5) *$x \neq x_1\langle_k\rangle_k x_2$, for any $\langle_k, \rangle_k \in \Omega_i$ and $x_1, x_2 \in E_i(\Sigma)^*$, i.e. there is no* empty parenthesization *in x.*

(6) *Suppose that $|x| = n$ and $x = x_1 \ldots x_n$, where $x_j \in \Sigma$, $(1 \le j \le n)$. Then either $n = 1$ and $x \in \Sigma$, or $n > 2$ and for all $1 \le j \le n$ we have*

$$x_j \in \{\bullet, \circ\} \Leftrightarrow 2 \le j \le n-1, x_{j-1} \in \Sigma \cup \Omega_{i,\mathrm{cl}}, \text{ and } x_{j+1} \in \Sigma \cup \Omega_{i,\mathrm{op}},$$

i.e. the operation symbols are placed in the correct positions.

(7) *If we set* $O_H = \{\, j \mid 1 \le j \le |x|, x_j = \bullet \,\}$, $\quad O_V = \{\, j \mid 1 \le j \le |x|, x_j = \circ \,\}$,

$$O_E = \{\, j \mid 1 \le j \le |x|, x_j \in \{\bullet, \circ\} \text{ and } \mathrm{dp}_x(j) \text{ is even} \,\},$$

$$O_O = \{\, j \mid 1 \le j \le |x|, x_j \in \{\bullet, \circ\},\ \textit{and}\ \mathrm{dp}_x(j)\ \textit{is odd}\},$$

then $\{O_H, O_V\} = \{O_E, O_O\}$*, i.e. the operation symbols alternate according to the depth of the parenthesization.*

Proof sketch. The necessity of conditions (1)–(7) easily follows from the construction of the elements of $\mathrm{TM}_i(\Sigma)$ described earlier in Definition 2.15 and in the definition of i-terms. However, the proof of sufficiency is rather long and technical. One can use induction on the length of a word x and check how the conditions guarantee that x can be cut into shorter subwords (by the "outer" operation symbols), in such a way that each segment also satisfies all the conditions except (3). This task will be left to the interested reader. Now Lemma 2.19 has the following corollary.

Lemma 2.20 *For all* $x = x_1 \dots x_n \in E_i(\Sigma)^+$ *we have* $x \in \mathrm{CTM}_i(\Sigma)$ *iff* $x \in \Sigma$*; or* $n > 2$*,* $x_1 \in \{\bullet, \circ\}$*,* $x_2 \dots x_n \in (\Sigma \cup \Omega_i)^+$ *and* $x_2 \dots x_n$ *satisfies conditions* (1)–(5) *of Lemma 2.19.*

One direct consequence of the above theorems is the following proposition, which will be used later on.

Proposition 2.21 $\mathrm{TM}_i(\Sigma)$ *and* $\mathrm{CTM}_i(\Sigma)$ *are deterministic context-free languages.*

Proof. It is not hard to construct a deterministic pushdown automaton which accepts the words that simultaneously satisfy conditions (1)–(3). Indeed, it a simple textbook exercise to draw a deterministic pushdown automaton that accepts Dyck_i or a slight modification of it, namely the language of the words that satisfy (1). As for (2), one can modify the previous machine so that if two opening parentheses follow each other, the automaton marks the second one when pushing it onto the stack, hence consecutive closing partners of these parentheses can be forbidden. The next modification for (3) is also straightforward: if the first letter is an opening parenthesis, push an extra symbol to the bottom of the stack, which can then be popped by reading any letter or operation symbol, but only after the first parenthesis has been closed.

Furthermore, conditions (4) to (7) can be described by regular languages, and it is well-known that deterministic context-free languages are closed under intersection with regular languages. In the case of $\mathrm{CTM}_i(\Sigma)$, the proof is similar. □

2.6.8 Properties of the Representations

Naturally the question arises of why we study so many biword representations. In the following we will briefly examine the advantages of using one rather than another.

The two-dimensional word representation (see Section 1.2) is the most intuitive one. It shows that biwords are really two-dimensional objects. It also accurately reflects the hierarchical relations in a biword.

Perhaps the easiest way of giving a biword is via the term representation. Furthermore, terms and operations between terms are very close to the various rational operations on binoid languages. The reader might recall that Hashiguchi et al. used the term representation to develop a theory of automata and grammars over binoid languages [HIJ00, HWJ03, HSJ04].

As we mentioned above, condensed terms are the input structures of parenthesizing automata, which will be introduced later. They directly confirm the sequential behavior of our automata. Moreover, they reduce the decidability algorithm of the membership problem of parenthesizing automata to an implementation level. Note that this is the only representation where the generalization to higher dimensions is nontrivial.

Trees are also widely used in computer science. It is worth realizing here that the biwords are, in fact, just ordered unranked trees whose leaves are labeled. This is quite similar to the representation of XML-documents by ordered unranked trees, but there the inner nodes are also labeled. Anyway, the tree representation might be a possible way of finding applications for our theory.

Sp-posets are essential for logical definability, and they are useful for exploring connections between biwords and "graph-like" structures, like 2-structures [ER90], texts [ER93, HtP96], pictures [GR97] and posets [LW00, Kus03a].

Next, we should mention that i-term and i-cterm representations are needed to relate our study to the results of Hashiguchi et al. Later in Section 3.11.3 we will also see that biwords can also be represented by alternating texts [HtP96].

2.7 Graph-Theoretic Characterizations

Several important classes of graphs can be characterized with the aid of forbidden subgraphs. Perhaps the best known example of this is Kuratowski's Theorem (see [Har94], for instance) which characterizes planar graphs. Besides this, its generalization known as the Robertson–Seymour Theorem [RS04, WikB], allows us to have characterizations by finite sets of forbidden graphs in a general setting.

Similar characterizations are known for sp-posets, series-parallel graphs and for reducible (m, n)-structures. Below we will briefly discuss each in turn. Moreover, as these results are not ours, the proofs will not be reproduced here. For such proofs the interested reader should consult the relevant literature [Val78, Gra81, VTL82, Gis88, Ési00].

Now let us consider the characterization of sp-posets. If we ignore the labeling aspect, posets can be identified with transitive directed acyclic graphs. We will provide the characterization in this setting. Directed acyclic graphs are usually abbreviated as DAG-s, and, of course, transitivity means that if (u, v) and (v, w) are edges of a transitive DAG, then (u, w) is an edge as well.

Definition 2.22 *We call an* "N"-digraph*, or "N" for short, the directed graph on* 4 *vertices,* $\{1, 2, 3, 4\}$*, whose edges are* $(1, 2)$*,* $(3, 2)$ *and* $(3, 4)$*. Similarly, we call a* P_4 graph *the undirected graph which consists of a path on four vertices.*

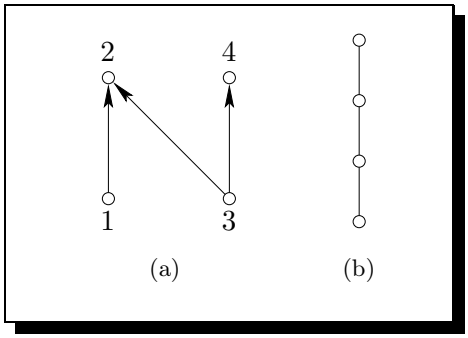

Figure 2.4: *The* N*-digraph (a) and the* P_4 *graph (b).*

These two graphs will play the role of forbidden structures in the characterizations. They are depicted in Figure 2.4. As usual, a subgraph G' of a graph G is called an *induced subraph* (or a *vertex induced subgraph*), if G' contains all edges of G that join two vertices which are present in G'.

Theorem 2.23 (**Grabowski [Gra81] and Valdes et al. [VTL82]**) *A poset is series-parallel iff it (viewed as a transitive DAG) does not contain an induced subgraph isomorphic to "N".*

In the words of [VTL82] "The proof of this fact is rather long ... The details can be found in [Val78] ..." Later a shorter proof was given by Gischer [Gis88]. A proof can also be found in [Kus03a].

Before giving the characterization of reducible (m, n)-structures we shall consider series-parallel graphs. Undirected graphs can be identified with $(0, 2)$-structures. Then the two associative and commutative operations correspond to the disjoint union and the join operations on graphs. (The joint of two graphs is the discrete union together with all the edges joining the vertices of the two graphs.) Those graphs that correspond

to reducible $(0,2)$-structures are called *series-parallel graphs*. Series-parallel graphs can also be defined as comparability graphs of series-parallel posets. More precisely, if $P = (P, <)$ is a poset, then its *comparability graph* is an undirected graph, whose set of vertices is P, and there is an edge between two vertices u and v iff $u < v$ or $v < u$ in P. Now a graph is series-parallel iff it is a comparability graph of a poset, which is series-parallel. Note that P_4 is not a series-parallel graph because both of its transitive orientations are isomorphic to the N-digraph. Series-parallel graphs are also called *complement-reducible graphs* (or *cographs* for short), as they are precisely those graphs which can be constructed from singleton graphs by the operations of complementation and disjoint union [CLB81]. Several other equivalent definition of cographs are also known (cf. [ISGCI, BLS99]). The forbidden structure characterization of the class is as follows.

Theorem 2.24 (**Cornel et al. [CLB81]**) *An (undirected) graph is a series-parallel graph iff it does not contain a P_4 as an induced subgraph.*

Actually, a common generalization of the above two theorems for reducible (m,n)-structures was given by Ésik.

We also say that a directed graph satisfies the N-condition if it has no induced subgraph isomorphic to "N". Similarly, an undirected graph satisfies the P_4-condition if it has no induced subgraph isomorphic to P_4. If $P = (P, <_1, \ldots, <_m, \sim_1, \ldots, \sim_n, \lambda)$ is an (m,n)-structure then we can define

$$\rho_i \quad := \quad \begin{cases} <_i \cup >_i & \text{if } 1 \leq i \leq m, \\ \sim_{i-m} & \text{if } m+1 \leq i \leq m+n. \end{cases}$$

Now the characterization of the reducible (m,n)-structures is the following.

Theorem 2.25 (**Theorem 2.14 of [Ésik00]**) *A nonempty (m,n)-structure $P = (P, <_1, \ldots, <_m, \sim_1, \ldots, \sim_n)$ is reducible iff the following conditions hold:*

1. *For each $i \in [m]$, the poset $(P, <_i)$ satisfies the N-condition.*
2. *For each $j \in [n]$, the graph $(P, \sim_j)$ satisfies the P_4-condition.*
3. *There exist no distinct vertices x, y, z and distinct integers $i, j, k \in [m+n]$ with $x \rho_i y$, $y \rho_j z$ and $z \rho_k x$. (The* triangle condition*).*

Since n-posets and biposets are special cases of (m,n)-structures, we also obtain the following corollaries. Note that – by definition – (m,n)-structures are total (i.e. each

pair of elements are related by exactly one of the relations), while n-posets and biposets may not have this property. On the other hand totality is clearly required for being constructible n-poset or series-parallel biposet.

Corollary 2.26 (**Theorem 2.1 of [ÉN04]**) *An n-poset $P = (P, <_1, \ldots, <_n, \lambda_P)$ is constructible iff the following conditions hold:*

1. *P is total, i.e. for every $x, y \in P$ with $x \neq y$ there is exactly one $i \in [n]$ such that $x <_i y$ or $y <_i x$ holds.*
2. *Each poset $(P, <_i)$, $i \in [n]$ satisfies the N-condition.*
3. *If x, y, z are different vertices of P, then x, y, z are related by at most 2 of the partial orders $<_i$. (The triangle condition).*

Corollary 2.27 *A biposet $P = (P, <_h, <_v, \lambda)$ is series-parallel iff the following conditions hold:*

1. *P is total, i.e. for all $x, y \in P$, with $x \neq y$ the vertices x and y are related either by $<_h$ or by $<_v$.*
2. *Both posets $(P, <_h)$ and $(P, <_v)$ satisfy the N-condition.*

Chapter 3

Parenthesizing Automata

In this chapter, an accepting device called parenthesizing automaton will be introduced to define the class of regular binoid languages. Since in general for models more complex than words the existence of an appropriate concept of automaton is far from being obvious, presenting a suitable concept of automaton for biwords is one of the main contributions of the thesis. The suitability of our model will be verified by the fact that the language class accepted by our automata (Reg) coincides with both of the classes of algebraically recognizable (Rec), and in monadic second-order logic definable (MSO) classes of binoid languages. Note that the latter classes exist and are well established, independently of the automaton concept chosen.

3.1 The Concept of Parenthesizing Automata

In this section we will introduce the concept of parenthesizing automata discussed in [ÉN04, Ném04, Ném06]. (The definition first appeared in [ÉN02].) Parenthesizing automata will be used to determine the class of regular binoid languages.

Recall that the term representation of a biword is an ordinary word over the extended alphabet $E(\Sigma) = \Sigma \cup \{ \bullet, \circ, \langle, \rangle \}$. In addition, in the term representation the usual rules of parenthesization are applied, but no superfluous parentheses are permitted.

At first sight it might be thought that a parenthesizing automaton reads term representations of biwords from left to right. When the automaton encounters a parenthesis symbol it performs a special, so-called parenthesizing transition. This transition is similar to ordinary transitions of classical finite automata, as it takes the automaton into another state, but parenthesizing transitions are labeled by parenthesis symbols instead of letters from Σ. The automaton is allowed to have several different pairs of parenthe-

sis symbols. But we require the following. In the term representation the opening and closing parentheses symbols have a well-defined standard matching, and the runs of the automaton must follow this matching. Namely, for each matching pair of parentheses in the term representation, the automaton has to perform such parenthesizing transitions whose labels also match, i.e. the first transition contains an opening parenthesis and the second one has the corresponding closing parenthesis.

On the other hand, automata do not perform transitions on the operation symbols, $\bullet$ and $\circ$. Instead the set of states S is divided into two disjoint subsets H and V. They are called horizontal and vertical states, respectively. Informally, when the automaton goes through a horizontal state, then it "applies" horizontal product, while crossing a vertical state implies a vertical product of the labels processed before and after that state.

From this also follows that each parenthesizing transition must lead from a horizontal state to a vertical one, or from a vertical state to a horizontal one. Hence the effect of parenthesizing transitions is to change the "actual" (horizontal/vertical) type of the operation which is used to connect the letters that are processed.

Definition 3.1 ([ÉN04]) *A (nondeterministic)* parenthesizing automaton*, PA for short, is a 9-tuple* $\mathcal{A} := (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$*, where* S *is a nonempty, finite set of* states*;* H *and* V *are the sets of* horizontal *and* vertical states *which give a disjoint partition of* S*,* Σ *is the* input alphabet *and* Ω *is a finite set of parentheses. Furthermore,*

- $\delta \subseteq (H \times \Sigma \times H) \cup (V \times \Sigma \times V)$ *is the* labeling transition relation*,*
- $\gamma \subseteq (H \times \Omega \times V) \cup (V \times \Omega \times H)$ *is the* parenthesizing transition relation*, and*
- $I, F \subseteq S$ *are the sets of* initial *and* final states*, respectively.*

In the following we will write $\mathrm{Type}(q) = \bullet$ if q is a horizontal state, and $\mathrm{Type}(q) = \circ$ if q is a vertical state of an automaton. Note that this definition has a parallel with the notion of types of biwords.

Example 3.2 A simple illustration of a PA is given in Figure 3.1. The horizontal states are those labeled by H_i and the vertical states are those labeled by V_j, for some i and j. There is a single initial state H_1, and a single final state H_7. The labeling transitions are

$$\delta = \{\, (H_1, a, H_2),\ (V_1, b, V_2),\ (H_3, c, H_4),\ (H_4, d, H_5),\ (H_6, e, H_7)\,\},$$

and the parenthesizing transitions are

$$\gamma = \{\, (H_2, \langle_1, V_1),\ (V_2, \langle_2, H_3),\ (H_5, \rangle_2, V_3),\ (V_3, \rangle_1, H_6)\,\}.$$

Later when we define the notion of run formally, we will see that this automaton has a single run from H_1 to H_7, hence the automaton just accepts the biword $a \bullet \langle b \circ \langle c \bullet d \rangle \rangle \bullet e$. Of course, if the automaton had cycles, the accepted binoid language would be more complicated than in our example. Several other examples of PA will be given later in Section 3.3.

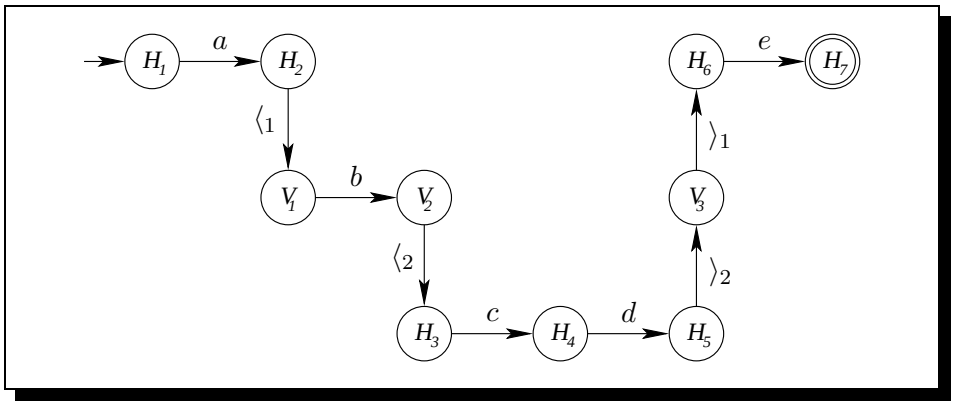

Figure 3.1: *A PA accepting* $\{ a \bullet \langle b \circ \langle c \bullet d \rangle \rangle \bullet e \}$.

3.2 The Operation of Parenthesizing Automata

In this section, we will define the operation of parenthesizing automata formally, in three different but equivalent ways. For this, let $[p, w, q]_{\mathcal{A}}$ denote that fact that automaton $\mathcal{A}$ can read the biword w from state p, and after processing w the automaton can be in state q. (Note that automata are nondeterministic, so it is possible that several $[p, w, q]_{\mathcal{A}}$ hold for the same p and w, but for different q-s.) If $[p, w, q]_{\mathcal{A}}$ holds, we also say that $\mathcal{A}$ *has a run on* w from p to q. For this notion three definitions will be given in turn, denoted by $[p, w, q]^1_{\mathcal{A}}$, $[p, w, q]^2_{\mathcal{A}}$ and $[p, w, q]^3_{\mathcal{A}}$, respectively. After that, in Section 3.2.4 we will establish their equivalence. In Section 3.2.5 we will define the class of regular binoid languages.

3.2.1 The First Approach

The very first definition for the operation of PA was presented in [ÉN02]. In this case we do not define what a run is, only the phrase "an automaton has a run on a biword w from state p to state q". The definition is based on an induction on the construction of w. It stresses the different functionality of the horizontal and vertical states and it also highlights the role of the parentheses.

Definition 3.3 (cf. Definition 3.3 of [ÉN04]) *Suppose that* $w \in \Sigma^*(\bullet, \circ)$ *and* $p, q \in S$. *We say that* $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ has a run on w from p to q, *denoted by* $[p, w, q]^1_{\mathcal{A}}$ *if one of the following conditions holds:*

(E) $w = \varepsilon$ *and* $p = q$.

(S) $w = a \in \Sigma$ *and* $(p, a, q) \in \delta$.

(HH) $p, q \in H$ *and* w *has a maximal horizontal decomposition* $w = w_1 \bullet \ldots \bullet w_n$, *where* $n \geq 2$, *and there exist* $r_1, \ldots, r_{n-1} \in S$, $r_0 = p$, $r_n = q$ *such that* $[r_{i-1}, w_i, r_i]^1_{\mathcal{A}}$, *for all* $i \in [n]$.

(VV) $p, q \in V$ *and* w *has a maximal vertical decomposition* $w = w_1 \circ \ldots \circ w_n$, *where* $n \geq 2$, *and there exist* $r_1, \ldots, r_{n-1} \in S$, $r_0 = p$, $r_n = q$ *such that* $[r_{i-1}, w_i, r_i]^1_{\mathcal{A}}$ *for all* $i \in [n]$.

(HV) $p, q \in H$ *and* w *has a maximal vertical decomposition* $w = w_1 \circ \ldots \circ w_n$, *where* $n \geq 2$, *and there exist* $\langle_k, \rangle_k \in \Omega$, $p', q' \in V$ *and* $(p, \langle_k, p'), (q', \rangle_k, q) \in \gamma$ *such that* $[p', w, q']^1_{\mathcal{A}}$ *holds.*

(VH) $p, q \in V$ *and* w *has a maximal horizontal decomposition* $w = w_1 \bullet \ldots \bullet w_n$, *where* $n \geq 2$, *and there exist* $\langle_k, \rangle_k \in \Omega$, $p', q' \in H$ *and* $(p, \langle_k, p'), (q', \rangle_k, q) \in \gamma$ *such that* $[p', w, q']^1_{\mathcal{A}}$ *holds.*

Definition 3.4 *Based on the above cases we introduce the following terminology. We say that, in case* (E) $\mathcal{A}$ *has an* empty run, *in case* (S) $\mathcal{A}$ *has a* singleton run, *in cases* (HH) *and* (VV) $\mathcal{A}$ *has a* direct run, *while in cases* (HV) *and* (VH) $\mathcal{A}$ *has an* indirect run *between* p *and* q.

Remark 3.5 Note that runs are always between states of the same type. Thus $[p, w, q]^1_{\mathcal{A}}$ implies $p, q \in H$ or $p, q \in V$. So in (HH) we have $r_1, r_2, \ldots, r_{n-1} \in H$, and similarly, in (VV) we have $r_1, r_2, \ldots, r_{n-1} \in V$. These observations will be frequently used in inductive proofs.

Example 3.6 The automaton of Example 3.2, depicted in Figure 3.1, accepts the singleton language which just consists of the biword $w = a \bullet \langle b \circ \langle c \bullet d \rangle \rangle \bullet e$. Indeed, there are runs $[H_1, a, H_2]_{\mathcal{A}}$, $[H_2, b \circ \langle c \bullet d \rangle, H_6]_{\mathcal{A}}$ and $[H_6, e, H_7]_{\mathcal{A}}$ corresponding to the horizontal decomposition of P. Moreover, the second run $[H_2, b \circ \langle c \bullet d \rangle, H_6]_{\mathcal{A}}$ exists, since it can be built from a parenthesizing transition $(H_2, \langle_1, V_1)$, a run $[V_1, b \circ \langle c \bullet d \rangle, V_3]_{\mathcal{A}}$ and a transition $(V_3, \rangle_1, H_6)$ according to the case (HV). Finally, the existence of the run $[V_1, b \circ \langle c \bullet d \rangle, V_3]_{\mathcal{A}}$ can be derived from the existence of the runs $[V_1, b, V_2]_{\mathcal{A}}$ and $[V_2, c \bullet d, V_3]_{\mathcal{A}}$, and the existence of the latter can be verified similarly using the case (VH).

3.2.2 The Second Approach

This second definition first appeared in [Ném05] and later in its journal version [Ném06]. Here we define the notion of run as certain finite sequences of transitions of an automaton. Recall that Ω is a finite set of parentheses that is partitioned into Ω_{op} of opening and Ω_{cl} of closing parentheses.

Let $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ be a PA. If $t = (p, x, q)$ is a labeling or parenthesizing transition of $\mathcal{A}$, i.e. $t \in \delta \cup \gamma$, then the *starting* and the *ending state* of t will be denoted by $\mathrm{start}(t) := p$ and $\mathrm{end}(t) := q$, respectively. Two transitions t_1 and t_2 are *adjacent* (in this order) if $\mathrm{end}(t_1) = \mathrm{start}(t_2)$. A words from $(\delta \cup \gamma)^*$ will be called *transition sequences*, but we will demand that in any transition sequence the consecutive transitions be adjacent. If $\mathbf{r} = t_1 t_2 \dots t_n \in (\delta \cup \gamma)^*$ is a transition sequence, then let $\mathrm{start}(\mathbf{r}) := \mathrm{start}(t_1)$ and $\mathrm{end}(\mathbf{r}) := \mathrm{end}(t_n)$. Here we say that two parenthesizing transitions $t_1 = (p, \omega_1, q)$ and $t_2 = (s, \omega_2, t) \in \gamma$ form a *parenthesizing transition pair* if ω_1 is an opening parenthesis and ω_2 is its closing partner. The concatenation of two transition sequences $\mathbf{r_1}$ and $\mathbf{r_2}$ will be denoted by $\mathbf{r_1 r_2}$, as usual.

Definition 3.7 ([Ném05]) *Let $\mathcal{A}$ be a PA. The set of its* runs, $\mathrm{Runs}(\mathcal{A})$, *is the least set of transition sequences that contains*

(i) *the* singleton runs: (p, σ, q), *for all* $(p, \sigma, q) \in \delta$;

(ii) *the* direct runs: $\mathbf{r_1 r_2}$, *for every* $\mathbf{r_1}, \mathbf{r_2} \in \mathrm{Runs}(\mathcal{A})$ *with* $\mathrm{end}(\mathbf{r_1}) = \mathrm{start}(\mathbf{r_2})$;

(iii) *the* indirect runs: $t_1 \mathbf{r}\, t_2$, *for every direct run* $\mathbf{r} \in \mathrm{Runs}(\mathcal{A})$, *and parenthesizing transition pair* t_1, t_2 *with* $\mathrm{end}(t_1) = \mathrm{start}(\mathbf{r})$ *and* $\mathrm{end}(\mathbf{r}) = \mathrm{start}(t_2)$.

Note that the empty sequence that would correspond to the empty run is not allowed. The reason for this is that it would be difficult trying to define the starting and the ending states of an empty run.

Suppose that $\mathcal{A}$ is a PA and $\mathbf{r} = t_1 \dots t_n \in \mathrm{Runs}(\mathcal{A})$. A parenthesizing transition pair t_i, t_j, $(i < j)$ is said to be a *matching parenthesizing transition pair in* $\mathbf{r}$ if $t_i \dots t_j$ is an indirect run of $\mathcal{A}$. Note that not every parenthesizing transition pair t_i, t_j, $(i < j)$ is a matching parenthesizing transition pair in $\mathbf{r}$. For instance, consider the first and the last transition in

$$(p_1, \langle, p_2) \dots (p_3, \rangle, p_4) \dots (p_5, \langle, p_6) \dots (p_7, \rangle, p_8),$$

where $\langle$ and $\rangle$ do not appear in other transitions.

It is obvious that every run of $\mathcal{A}$ is of the form

$$\mathbf{r} = t_1 t_2 t_3 \dots t_n = (p_0, \omega_1, p_1)(p_1, \omega_2, p_2)(p_2, \omega_3, p_3) \dots (p_{n-1}, \omega_n, p_n),$$

where $p_i \in S$ and $\omega_i \in \Sigma \cup \Omega$ for all $i = 1, \dots, n$. If $\mathbf{r}$ is an indirect run, then t_1 and t_n is a matching parenthesizing transition pair, and $t_2 \dots t_{n-1}$ is a direct run of $\mathcal{A}$. Moreover, if $\mathbf{r}$ is a direct run, then it has a unique decomposition into subruns $\mathbf{r} = \mathbf{r}_1 \mathbf{r}_2 \dots \mathbf{r}_k$, where each $\mathbf{r}_i$ is either a singleton run or an indirect run for $i = 1, \dots, k$, and $k \geq 2$.

Definition 3.8 *Suppose that $\mathcal{A}$ is a PA and $\mathbf{r} \in \mathrm{Runs}(\mathcal{A})$. The* label *of $\mathbf{r}$ is a biword from $\Sigma^+(\bullet, \circ)$ defined inductively as follows:*

(i) *If $\mathbf{r} = (p, \sigma, q)$, then $\mathrm{Label}(\mathbf{r}) := \sigma$.*

(ii) *If $\mathbf{r}$ is a direct run, and $\mathbf{r} = \mathbf{r_1} \mathbf{r_2}$ for some $\mathbf{r_1}, \mathbf{r_2} \in \mathrm{Runs}(\mathcal{A})$, then*

- *if $\mathrm{end}(\mathbf{r_1}) \in H$, then $\mathrm{Label}(\mathbf{r}) := \mathrm{Label}(\mathbf{r_1}) \bullet \mathrm{Label}(\mathbf{r_2})$;*
- *if $\mathrm{end}(\mathbf{r_1}) \in V$, then $\mathrm{Label}(\mathbf{r}) := \mathrm{Label}(\mathbf{r_1}) \circ \mathrm{Label}(\mathbf{r_2})$.*

(iii) *If $\mathbf{r}$ is an indirect run $\mathbf{r} = t_1 \mathbf{r}' t_2$, then $\mathrm{Label}(\mathbf{r}) := \mathrm{Label}(\mathbf{r}')$.*

Since $\bullet$ and $\circ$ are associative, the definition of $\mathrm{Label}(\mathbf{r})$ does not depend on the choice of factorization in case (ii) above.

This approach has the advantage that we can describe those transition sequences that are runs by a few simple rules.

Lemma 3.9 *For every nonempty transition sequence $\mathbf{r}$ of an automaton $\mathcal{A}$*

$$\mathbf{r} \in \mathrm{Runs}(\mathcal{A}) \text{ iff } \mathbf{r} \text{ obeys the four rules below.}$$

***Rule 1**: In $\mathbf{r}$ the use of parentheses is syntactically correct, i.e. if we concatenate the middle component of the transitions, we get a string in which all parentheses are matched.*

***Rule 2**: There is no double-parenthesization in $\mathbf{r}$, i.e. there is no subsequence of the form*

$$\dots (p_i, \langle_k, p_{i+1})(p_{i+1}, \langle_l, p_{i+2})\, \mathbf{r}'\, (p_j, \rangle_l, p_{j+1})(p_{j+1}, \rangle_k, p_{j+2}) \dots,$$

where in $\mathbf{r}'$ the use of parentheses is balanced.

***Rule 3**: No singleton is parenthesized in $\mathbf{r}$, i.e.*

$$\dots (p_i, \langle_k, p_{i+1})(p_{i+1}, \sigma, p_{i+2})(p_{i+2}, \rangle_k, p_{i+3}) \dots$$

is forbidden.

***Rule 4**: There is no empty parenthesization in* $\mathbf{r}$*, i.e.*

$$\dots(p_i, \langle_k, p_{i+1})(p_{i+1}, \rangle_k, p_{i+2})\dots$$

is forbidden.

Proof sketch. The necessity of the rules can be seen by induction on the structure of the run $\mathbf{r}$. In a similar way their sufficiently can be verified using induction on the length of the transition sequence $\mathbf{r}$. Both proofs apply the fact that a run of $\mathcal{A}$ is an indirect run if and only if its first and last transition is a matching parenthesizing transition pair.

Now, it is straightforward to define $[p, P, q]^2_{\mathcal{A}}$ with the help of $\mathrm{Runs}(\mathcal{A})$.

Definition 3.10 *Let* $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ *be a PA,* $p, q \in S$ *and* $w \in \Sigma^*(\bullet, \circ)$*. Let us write* $[p, w, q]^2_{\mathcal{A}}$ *if* $w = \varepsilon$ *and* $p = q$*; or if there is a run* $\mathbf{r} \in \mathrm{Runs}(\mathcal{A})$ *with* $\mathrm{start}(\mathbf{r}) = p$*,* $\mathrm{end}(\mathbf{r}) = q$*, and* $\mathrm{Label}(\mathbf{r}) = w$*.*

3.2.3 The Third Approach

Obviously the formulation of the concept of automata can be based on any representation of biwords that were studied in the previous chapter. While the first and the second approaches are closest to the term representation, our third approach is directly based on the condensed term representation. Furthermore, this approach is more formal. Given a condensed term as input the overall state of the computation at any given moment is described by a configuration. As usual, we define a transition relation on the set of configurations. This approach emphasizes the sequential nature and the algorithmic feature of the behavior of our automata, hence it is closest to the implementation level. This approach has not yet appeared in publications.

Definition 3.11 *The set of* configurations *of a PA* $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ *is*

$$\mathrm{Conf}(\mathcal{A}) = S \times \Omega^*_{op} \times \Big(\Sigma \cup \{\, \langle, \rangle \,\}\Big)^* .$$

Let w^{ctm} be a condensed term representation of a biword $w \in \Sigma^*(\bullet, \circ)$. Recall that w^{ctm-} denotes w^{ctm} without its type-sign (the first letter).

During a computation of an automaton $\mathcal{A}$ on w^{ctm-} the first component of the configuration is the actual state of the automaton. The second is the sequence of parentheses that are opened but to yet closed. The third component is the sequence of remaining letters of the input, i.e. a suffix of w^{ctm-}.

Now the *transition relation* on the configurations can be described as follows. For any $q, q' \in S$, $\alpha, \alpha' \in \Omega_{op}^*$, $\sigma \in \Sigma \cup \{\langle, \rangle\}$ and a suffix w of a condensed term we have

$$(q, \alpha, \sigma w) \vdash (q', \alpha', w), \text{ iff } \begin{cases} \sigma \in \Sigma, & \alpha' = \alpha, & \text{and} & (q, \sigma, q') \in \delta; \text{ or} \\ \sigma = \langle, & \alpha' = \alpha\langle_i, & \text{and} & (q, \langle_i, q') \in \gamma; \text{ or} \\ \sigma = \rangle, & \alpha'\langle_i = \alpha, & \text{and} & (q, \rangle_i, q') \in \gamma. \end{cases}$$

As usual, let $\vdash^*$ denote the reflexive transitive closure of $\vdash$.

Recall that we write $\mathrm{Type}(p) = \bullet$, if p is a horizontal state, while $\mathrm{Type}(p) = \circ$ means that p is a vertical state. Similarly $\mathrm{Type}(w) = \bullet$ and $\mathrm{Type}(w) = \circ$ indicate that w is a horizontal/vertical biword. Finally, we set

$$[p, w, q]^3_{\mathcal{A}} \Leftrightarrow \begin{cases} (p, \varepsilon, w^{ctm-}) \vdash^* (q, \varepsilon, \varepsilon) & \text{if } w \in \{\varepsilon\} \cup \Sigma \text{ or } \mathrm{Type}(p) = \mathrm{Type}(w); \\ (p, \varepsilon, \langle w^{ctm-}\rangle) \vdash^* (q, \varepsilon, \varepsilon) & \text{if } w \notin \{\varepsilon\} \cup \Sigma \text{ and } \mathrm{Type}(p) \neq \mathrm{Type}(w). \end{cases}$$

From the definitions above it is obvious that the second component of a configuration works as a stack storage.

3.2.4 Equivalence the Three Approaches

First of all, it is not hard to see that the phrase "automaton $\mathcal{A}$ has a direct (indirect, singleton) run on biword w" of Definition 3.4 is in accordance with the term "$\mathbf{r}$ is a direct (indirect, singleton, resp.) run" of Definition 3.7. Next, we will formulate some basic properties of the second approach.

If $\mathbf{r} = (p_0, \omega_1, p_1)(p_1, \omega_2, p_2)\dots(p_{n-1}, \omega_n, p_n)$ is a run of $\mathcal{A}$, we can define the *word* of $\mathbf{r}$ as

$$\mathrm{Word}(\mathbf{r}) := \omega'_1\omega'_2\dots\omega'_n,$$

where

$$\omega'_i := \begin{cases} \omega_i & \text{if } \omega_i \in \Sigma, \\ \langle & \text{if } \omega_i \in \Omega \text{ is an opening parenthesis, and} \\ \rangle & \text{if } \omega_i \in \Omega \text{ is a closing parenthesis.} \end{cases}$$

Thus $\mathrm{Word}(\mathbf{r})$ is no other than the concatenation of the middle components of the transitions in $\mathbf{r}$, after the indices of the parentheses have been deleted.

The relationship between the label and the word of a run is given by the following lemma.

Lemma 3.12 *Suppose that $\mathcal{A}$ is a PA, $\mathbf{r} \in \mathrm{Runs}(\mathcal{A})$, and $w = \mathrm{Label}(\mathbf{r})$. Then the following statements hold.*

(i) $\mathbf{r}$ *is a singleton run* $\Leftrightarrow$ $w \in \Sigma$ $\Leftrightarrow$ $\mathrm{Word}(\mathbf{r}) = w^{ctm}$.

(ii) $\mathbf{r}$ *is a direct run* $\Leftrightarrow$ $\mathrm{Type}(\mathrm{start}(\mathbf{r})) = \mathrm{Type}(w)$ $\Leftrightarrow$ $\mathrm{Word}(\mathbf{r}) = w^{ctm-}$.

(iii) $\mathbf{r}$ *is indirect run* $\Leftrightarrow$ $\mathrm{Type}(\mathrm{start}(\mathbf{r})) \neq \mathrm{Type}(w)$ $\Leftrightarrow$ $\mathrm{Word}(\mathbf{r}) = \langle w^{ctm-} \rangle$.

Proof sketch. These statements are straightforward consequences of the earlier definitions. Simply use induction on the construction of $\mathbf{r}$.

Now we are ready to prove the equivalence of our three approaches.

Lemma 3.13 *If $\mathcal{A}$ is a PA, $p, q \in S$ and $w \in \Sigma^*(\bullet, \circ)$, then*

$$[p, w, q]^1_{\mathcal{A}} \Leftrightarrow [p, w, q]^2_{\mathcal{A}} \Leftrightarrow [p, w, q]^3_{\mathcal{A}}.$$

Proof. The equivalence $[p, w, q]^2_{\mathcal{A}} \Leftrightarrow [p, w, q]^3_{\mathcal{A}}$ can be proven by Lemma 3.12. One should distinguish between the cases of singletons, $\mathrm{Type}(p) = \mathrm{Type}(w)$ and $\mathrm{Type}(p) \neq \mathrm{Type}(w)$. Alternatively, we can prove both equivalences by induction on the construction of w. For this, we should verify that both in the second definition and in the third definition the runs can be decomposed into subruns – according to the decomposition of w – exactly as in the cases (E), (S), (HH), (HV) (VV), and (VH) of the first definition.

The cases (E) and (S) are trivial, all $[p, w, q]^1_{\mathcal{A}}$, $[p, w, q]^2_{\mathcal{A}}$ and $[p, w, q]^3_{\mathcal{A}}$ are equivalent to $p = q$ (in case (E)) or to $(p, w, q) \in \delta$ (in case (S)).

Here we just prove the equivalence of $[p, w, q]^1_{\mathcal{A}}$, $[p, w, q]^2_{\mathcal{A}}$ in the case (HH). Hence suppose that $p, q \in H$ and $w \in \Sigma^*(\bullet, \circ)$ is a horizontal biword, with maximal horizontal decomposition $w = w_1 \bullet \ldots \bullet w_n$, $(n \geq 2)$. We need to show that $[p, w, q]^2_{\mathcal{A}}$ iff there exist $p_1, \ldots, p_{n-1} \in S$, $p_0 = p$, $p_n = q$ such that $[p_{i-1}, w_i, p_i]^2_{\mathcal{A}}$, for all $i \in [n]$.

By Definition 3.7 and Definition 3.10, the condition above is sufficient for $[p, w, q]^2_{\mathcal{A}}$.

To prove that the converse applies, if $[p, w, q]^2_{\mathcal{A}}$, then there exist an $\mathbf{r} \in \mathrm{Runs}(\mathcal{A})$ such that $\mathrm{start}(\mathbf{r}) = p, \mathrm{end}(\mathbf{r}) = q$, and $\mathrm{Label}(\mathbf{r}) = w$. Since $\mathrm{Type}(w) = \mathrm{Type}(p)$, by Lemma 3.12 $\mathbf{r}$ is a direct run. Now $\mathbf{r}$ can be uniquely written as $\mathbf{r} = \mathbf{r_1} \ldots \mathbf{r_m}$, where each $\mathbf{r_i}$ is an indirect or singleton run, for all $i \in [m]$. Furthermore, we have $w = \mathrm{Label}(\mathbf{r_1}) \bullet \ldots \bullet \mathrm{Label}(\mathbf{r_m})$. But the maximal horizontal decomposition of w is unique, hence $n = m$ and $\mathrm{Label}(\mathbf{r_i}) = w_i$, for all $i \in [n]$. Finally, setting $p_0 = p$, and $p_i = \mathrm{end}(\mathbf{r_i})$ for all $i \in [n]$, we have $[p_{i-1}, w_i, p_i]^2_{\mathcal{A}}$, for all $i \in [n]$, as required.

It can be proved in a similar way that $[p, w, q]^2_{\mathcal{A}}$ behaves exactly as $[p, w, q]^1_{\mathcal{A}}$ in the remaining cases (HV), (VV), and (VH) as well. The same holds for $[p, w, q]^3_{\mathcal{A}}$ too. $\square$

3.2.5 Acceptance and Regular Languages

As the equivalence of $[p,w,q]^1_{\mathcal{A}}$, $[p,w,q]^2_{\mathcal{A}}$ and $[p,w,q]^3_{\mathcal{A}}$ has now been established in Lemma 3.13, we will use the notation $[p,w,q]_{\mathcal{A}}$ without superscript for any of them.

A run from an initial state to a final state will be called an *accepting run*, and the binoid language accepted by a PA is defined as the set of labels of the accepting runs.

Definition 3.14 *The binoid* language $L(\mathcal{A})$ accepted by the parenthesizing automaton $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ *is defined as*

$$L(\mathcal{A}) \;:=\; \{\, w \in \Sigma^*(\bullet,\circ) \mid [i,w,f]_{\mathcal{A}} \text{ for some } i \in I \text{ and } f \in F \,\}.$$

Definition 3.15 *A binoid language $L \subseteq \Sigma^*(\bullet,\circ)$ is called* regular *if there exists a PA $\mathcal{A}$ that accepts it, i.e. $L = L(\mathcal{A})$. Let* $\mathsf{Reg}(\Sigma)$ *denote the* class of regular binoid languages *over Σ. Moreover, let* Reg *stand for the union of the classes* $\mathsf{Reg}(\Sigma)$ *for all Σ. Two automata are said to be* equivalent *if they accept the same language.*

Overall, we can summarize acceptance by direct and indirect runs. The acceptance of a nonempty and nonsingleton biword w can occur directly when the type of w is the same as the type of the initial and final state; or indirectly when the aforementioned types are different. The following fact summarizes these modes of acceptance.

Fact 3.16 *If $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ is a PA, and w is a horizontal biword, then*

$$\begin{aligned} w \in L(A) \Leftrightarrow \quad & \textit{either} \quad && \textit{i) } [i,w,f]_{\mathcal{A}} \text{ for some } i \in I \cap H \text{ and } f \in F \cap H; \\ & \textit{or} && \textit{ii) } (i,\langle,r),(s,\rangle,f) \in \gamma, \text{ and } [r,w,s]_{\mathcal{A}}, \text{ where } r,s \in H, \\ & && \quad \text{for some } i \in I \cap V,\ f \in F \cap V \text{ and } \langle,\rangle \in \Omega. \end{aligned}$$

An analogous statement holds for a vertical biword w:

$$\begin{aligned} w \in L(A) \Leftrightarrow \quad & \textit{either} \quad && \textit{i) } [i,w,f]_{\mathcal{A}} \text{ for some } i \in I \cap V, \text{ and } f \in F \cap V; \\ & \textit{or} && \textit{ii) } (i,\langle,r),(s,\rangle,f) \in \gamma \text{ and } [r,w,s]_{\mathcal{A}}, \text{ where } r,s \in V, \\ & && \quad \text{for some } i \in I \cap H,\ f \in F \cap H \text{ and } \langle,\rangle \in \Omega. \end{aligned}$$

Finally, for a singleton biword $\sigma \in \Sigma$ the type of the initial and final states can be both horizontal or vertical, thus

$$\sigma \in L(A) \Leftrightarrow [i,\sigma,f]_{\mathcal{A}}, \text{ where } i \in I \text{ and } f \in F.$$

3.3 Examples

In order to give a more intuitive picture about PA and regular binoid languages, here we will provide some examples.

Example 3.17 The language of all biwords over Σ, namely $\Sigma^*(\bullet, \circ)$, is regular. Indeed, the automaton with two states – depicted in Figure 3.2 – accepts it.

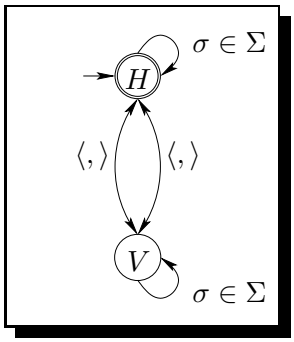

Figure 3.2: *An automaton accepting all biwords of* $\Sigma^*(\bullet, \circ)$.

Let us introduce for all $n \geq 0$ the following notation for exponentiations:

$$w^{\bullet n} := \underbrace{w \bullet w \bullet \ldots \bullet w}_{n \text{ times}} \quad \text{and} \quad w^{\circ n} := \underbrace{w \circ w \circ \ldots \circ w}_{n \text{ times}}.$$

Of course, we let $w^{\bullet 0} := w^{\circ 0} = \varepsilon$, the empty biword.

Example 3.18 Consider the following language

$$L = \{\, a \bullet \langle c^{\circ n} \rangle \bullet a \mid n \geq 2 \,\} \cup \{\, b \bullet \langle c^{\circ n} \rangle \bullet b \mid n \geq 2 \,\}.$$

Now, L is accepted by both automata shown in Figure 3.3. $\mathcal{A}_1$ has 8 states, while $\mathcal{A}_2$ has just 7. The difference comes from the fact that $\mathcal{A}_2$ has two pairs of parentheses, while $\mathcal{A}_1$ has only one. Hence $\mathcal{A}_2$ can store information about the first letter with the help of the index of the first parenthesis. Later in Section 3.8 we will see that this is an important feature of PA. If we were to restrict the number of parenthesis symbols that could be used in PA, the accepting capacity would also be restricted.

Example 3.19 The automaton shown in Figure 3.4 accepts the infinite language

$$\widehat{L} = \{\, c,\ a \bullet \langle b \circ c \rangle,\ a \bullet \langle b \circ \langle a \bullet \langle b \circ c \rangle \rangle \rangle,\ \ldots \,\}.$$

This language is the least solution of the fixed point equation

$$X = \{a\} \bullet (\{b\} \circ X) + \{c\}.$$

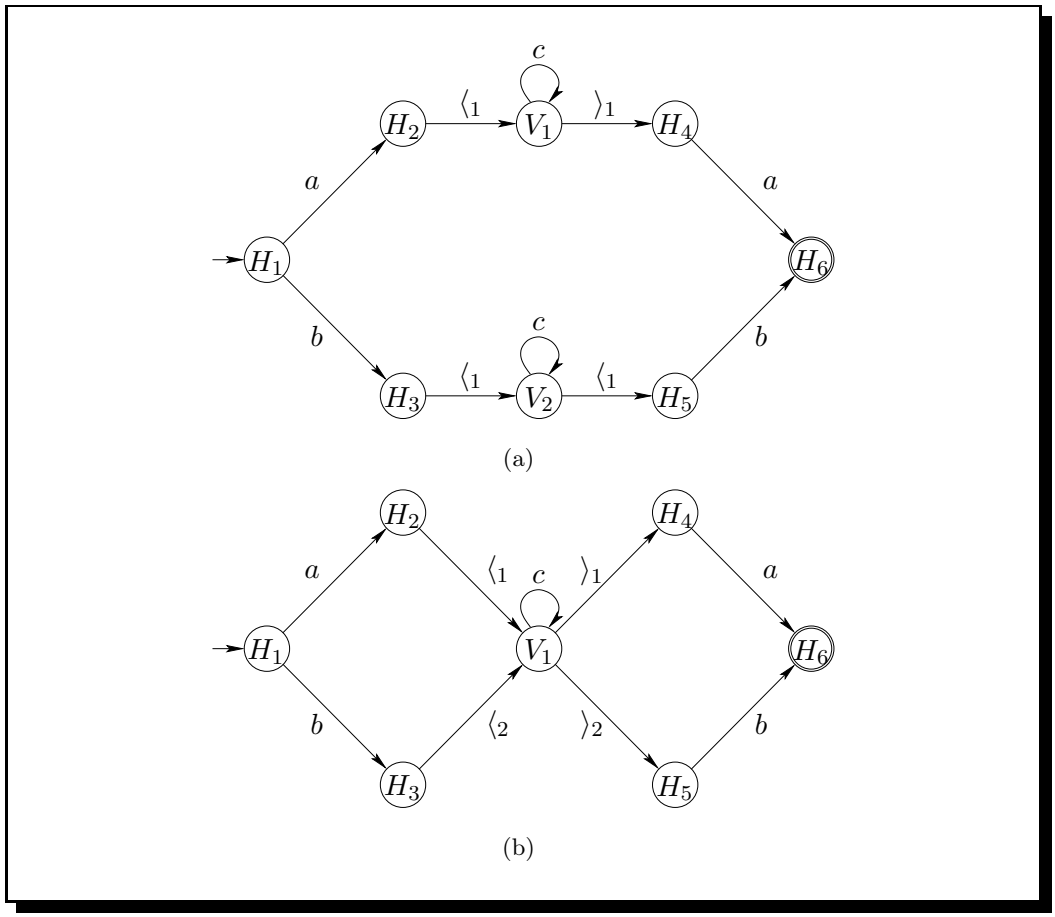

Figure 3.3: *Automata $\mathcal{A}_1$ (a) and $\mathcal{A}_2$ (b) which recognize the binoid language described in Example 3.18.*

(The biword operations $\bullet$ and $\circ$ extend to languages in a natural way. See Section 3.10 for details.) This example is interesting because it shows a regular language that has unbounded depth. This property means that we cannot give an upper bound for the number of nested parentheses used in the elements of $\widehat{L}$, (see Section 3.10 for the precise definition). Later we will find that the existence of such a language makes it hard to characterize the class Reg as languages obtained from the singletons by a few simple operations.

3.4 The Problem of Double-Parenthesization

By *double-parenthesization* in a transition sequence we mean two opening parenthesizing transitions right next to each other whose closing partners are also right next to each

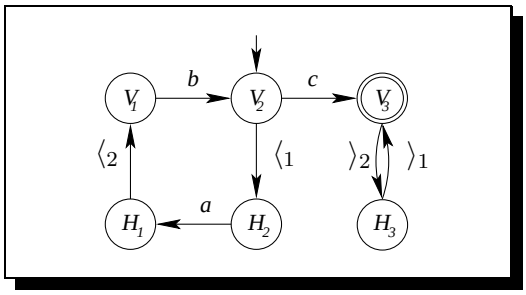

Figure 3.4: *An automaton accepting* $\widehat{L} = \{c, a \bullet \langle b \circ c\rangle, a \bullet \langle b \circ \langle a \bullet \langle b \circ c\rangle\rangle\rangle, \ldots\}$.

other. For instance, the sequence of transitions

$$(p, \langle_1, q)(q, \langle_2, r)(r, a, s)(s, b, t)(t, \rangle_2, u)(u, \rangle_1, v)$$

contains double-parenthesization. We usually abbreviate this sequences by the diagram

$$p \xrightarrow{\langle_1} q \xrightarrow{\langle_2} r \overset{w}{\dashrightarrow} t \xrightarrow{\rangle_2} u \xrightarrow{\rangle_1} v,$$

where $w = a \bullet b$ or $w = a \circ b$, depending on the types of the states. Note that, according to **Rule 2** of Lemma 3.9, such sequences are not valid runs.

If we allowed such double-parenthesization then one might use cycles of superfluous parentheses to count. Thus nonrecognizable languages like $\{a^{\bullet n} \bullet b \bullet c^{\bullet n} \mid n \geq 0\}$ could be accepted. See Figure 3.5.

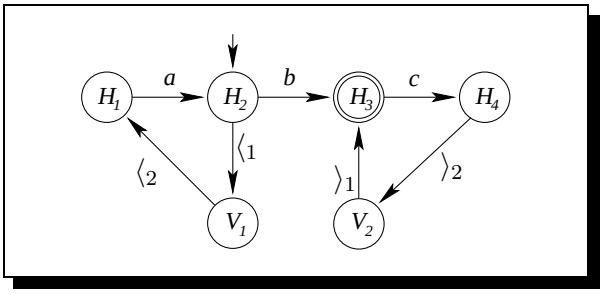

Figure 3.5: *This automaton accepts a recognizable language – namely* $\{b\}$ *– only if double-parenthesization is forbidden.*

After seeing the necessity of forbidding double-parenthesization, the next example makes it clear that this also makes it difficult to prove the closure of regular languages under substitution. Suppose that in an automaton $\mathcal{A}_1$ we want to substitute a transition (p, ξ, q) with all accepting paths of an automaton $\mathcal{A}_2$. Next, assume that p and q are horizontal states. If $\mathcal{A}_2$ has a vertical initial state i_v and a vertical final state f_v, then

it seems necessary to draw new transitions $(p, \langle_*, i_v)$ and $(f_v, \rangle_*, q)$, where $\langle_*, \rangle_*$ is a new pair of parentheses. But this is not enough in itself, as this construction does not ensure the simulation of indirect runs like

$$i_v \xrightarrow{\langle} s \overset{w}{\dashrightarrow} t \xrightarrow{\rangle} f_v$$

between p and q since

$$p \xrightarrow{\langle_*} i_v \xrightarrow{\langle} s \overset{w}{\dashrightarrow} t \xrightarrow{\rangle} f_v \xrightarrow{\rangle_*} q$$

is not allowed.

It is tempting to get around this problem by copying the transitions starting from s, but using p as the origin instead of s, and also copying the transitions arriving at t, but using q as the target instead of t.

But then it is possible that we have more runs than necessary. For instance, in $\mathcal{A}_1$

$$i_v \xrightarrow{\langle} s \xrightarrow{\langle'} s' \xrightarrow{w} t' \xrightarrow{\rangle'} t \xrightarrow{\rangle} f_v$$

is not allowed, but the construction above cuts the two outer parentheses, so

$$p \xrightarrow{\langle'} s' \xrightarrow{w} t' \xrightarrow{\rangle'} q$$

becomes a valid run. On the other hand, we cannot simply neglect the transitions $(s, \langle', s')$ and $(t', \rangle', t)$ since they may be part of other valid runs too. The next three sections will tell us how to overcome these difficulties.

3.5 The Substitution Product of Automata

The definition of substitution product of automata may look strange at the first sight, but it will be an adequate tool for proving normal form theorems later.

In the following, we shall assume that no automaton has two opening or closing parenthesizing transitions with the same label. This can easily be achieved by replacing the multiple occurrences of the same parenthesizing transition pair with new transitions using different symbols.

Definition 3.20 ([Ném06]) *Suppose that* $\mathcal{A}_1 = (S_1, H_1, V_1, \Sigma, \Omega, \delta_1, \gamma_1, I_1, F_1)$ *and* $\mathcal{A}_2 = (S_2, H_2, V_2, \Sigma, \Omega, \delta_2, \gamma_2, I_2, F_2)$ *are PA, and either* $p, q \in H_1$ *and* $R, S \subseteq H_2$*; or* $p, q \in V_1$ *and* $R, S \subseteq V_2$*. Next, further assume that* S_1 *and* S_2 *are disjoint. We define the* substitution product *of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *with respect to* p*,* q*,* R *and* S *as*

$$\mathcal{A}_1 *_{[p \to R, S \to q]} \mathcal{A}_2 := (S_3, H_3, V_3, \Sigma, \Omega_3, \delta_3, \gamma_3, I_1, F_1),$$

where

$$
\begin{aligned}
S_3 &:= S_1 \cup S_2, \quad H_3 := H_1 \cup H_2, \quad V_3 := V_1 \cup V_2, \\
\Omega_3 &:= \Omega \cup \{\, \langle^{\text{first}}, \rangle^{\text{first}},\ \langle^{\text{last}}, \rangle^{\text{last}} \ \mid\ \langle, \rangle \in \Omega \,\}, \\
\delta_3 &:= \delta_1 \cup \delta_2 \\
&\quad \cup \{\, (p, a, x) \mid a \in \Sigma, x \in S_2, \exists r \in R : (r, a, x) \in \delta_2 \,\} \\
&\quad \cup \{\, (y, b, q) \mid y \in S_2, b \in \Sigma, \exists s \in S : (y, b, s) \in \delta_2 \,\}, \\
\gamma_3 &:= \gamma_1 \cup \gamma_2 \\
&\quad \cup \{\, (p, \langle^{\text{first}}, x), (y, \rangle^{\text{first}}, z) \mid x, y, z \in S_2, \exists r \in R : (r, \langle, x), (y, \rangle, z) \ \in \gamma_2 \,\} \\
&\quad \cup \{\, (x, \langle^{\text{last}}, y), (z, \rangle^{\text{last}}, q) \mid x, y, z \in S_2, \exists s \in S : (x, \langle, y), (z, \rangle, s) \ \in \gamma_2 \,\}.
\end{aligned}
$$

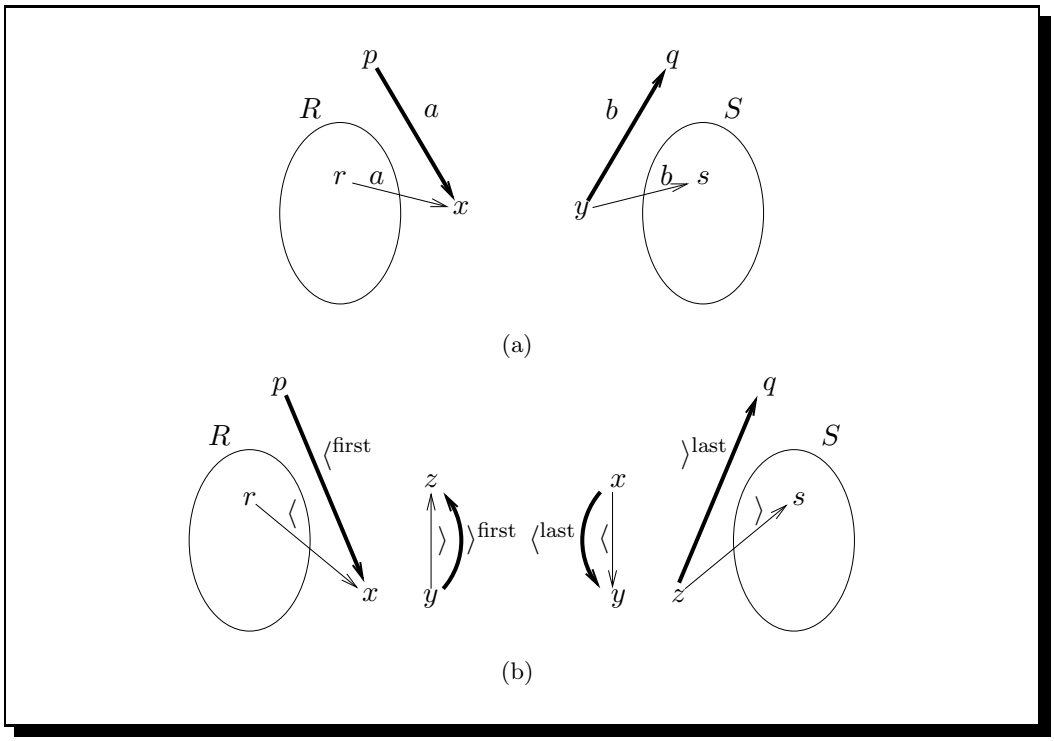

Figure 3.6: *The labeling (a) and the parenthesizing (b) transitions used in Definition 3.20. The thin arrows represent the original transitions, and the thick arrows the new ones.*

The construction is illustrated in Figure 3.6. If both R and S are singletons, say $R = \{r\}$ and $S = \{s\}$, then we can write $\mathcal{A}_1 *_{[p \to r, s \to q]} \mathcal{A}_2$ instead of $\mathcal{A}_1 *_{[p \to \{r\}, \{s\} \to q]} \mathcal{A}_2$. The next lemma formulates the key property of the substitution product. Namely, $\mathcal{A}_1 *_{[p \to R, S \to q]} \mathcal{A}_2$ from p to q has all the runs of $\mathcal{A}_2$ between two states from S and R which runs have the same type as p and q.

Lemma 3.21 ([Ném06]) *Suppose that $\mathcal{A}_1$ and $\mathcal{A}_2$ are PA as before, $p, q \in H_1$, $R, S \subseteq H_2$ and $\mathcal{A}_3 = \mathcal{A}_1 *_{[p\to R, S\to q]} \mathcal{A}_2$. Moreover, $p \neq q$, and no transition of $\mathcal{A}_1$ arrives at p or starts from q. Then for every $w \in \Sigma^*(\bullet, \circ)$*

$$[p,w,q]_{\mathcal{A}_3} \Leftrightarrow \quad \begin{array}{ll} \textit{either} & \textit{i)}\ [p,w,q]_{\mathcal{A}_1}, \\ \textit{or} & \textit{ii)}\ \exists r \in R, \exists s \in S\ :\ [r,w,s]_{\mathcal{A}_2}\ \textit{and } w\ \textit{is horizontal.} \end{array}$$

Proof. $[p,w,q]_{\mathcal{A}_3}$ implies that $w = \mathrm{Label}(\mathbf{r})$ for a run $\mathbf{r} = t_1 \dots t_m \in \mathrm{Runs}(\mathcal{A}_3)$ with $\mathrm{start}(t_1) = p$ and $\mathrm{end}(t_m) = q$.

If $\mathrm{end}(t_1) \in S_1$, then $\mathbf{r} \in \mathrm{Runs}(\mathcal{A}_1)$ also holds. This follows from the definition of $\mathcal{A}_3$ and from the fact that in $\mathcal{A}_1$ no transition arrives at p, hence $\mathbf{r}$ cannot leave the $\mathcal{A}_1$ component. Thus case (i) is true.

If $\mathrm{end}(t_1) \in S_2$, then we will be able to modify $\mathbf{r}$ to obtain a run $\mathbf{r}' := t'_1 \dots t'_m \in \mathrm{Runs}(\mathcal{A}_2)$ with $\mathrm{Label}(\mathbf{r}') = w$ and $\mathrm{start}(t'_1) \in R$, and $\mathrm{end}(t'_m) \in S$. Indeed, if t_1 is of the form $t_1 = (p, a, x)$, $a \in \Sigma$, then there is an $r \in R$ such that $t'_1 := (r, a, x) \in \delta_2$. Similarly, if $t_m = (y, b, q)$, $b \in \Sigma$, then there is an $s \in S$ such that $t'_m := (y, b, s) \in \delta_2$. On the other hand, if t_1 or t_m involves a parenthesis, like $t_1 = (p, \langle^{\mathrm{first}}, x)$, then there is a closing transition partner of t_1, say $t_i = (y, \rangle^{\mathrm{first}}, z)$, where $i < m$, and $x, y, z \in S_2$. Moreover, by definition, there is an $r \in R$ such that $t'_1 := (r, \langle, x)$, $t'_i := (y, \rangle, z) \in \gamma_2$. Similarly, if $t_m = (z, \rangle^{\mathrm{last}}, q)$, then there is an index $j > 1$ such that $t_j = (x, \langle^{\mathrm{last}}, y)$. So we can set $t'_j := (x, \langle, y)$ and $t'_m := (z, \rangle, s) \in \gamma_2$ for a suitable $s \in S$. So far we have defined t'_k for at most four k-s. Now let $t'_k := t_k$ for all other k-s (note that $t_k \in \delta_2 \cup \gamma_2$ in such cases). Now $\mathrm{Label}(\mathbf{r}') = w$, $\mathrm{start}(t'_1) = r \in R$, and $\mathrm{end}(t'_m) = s \in S$ implies $[r,w,s]_{\mathcal{A}_2}$. Since $\langle^{\mathrm{first}}$ and $\rangle^{\mathrm{last}}$ do not match, t_1 and t_m cannot be a matching parenthesizing transition pair. Hence $\mathbf{r}$ is a direct run, and so is $\mathbf{r}'$. Consequently, by Lemma 3.12, $r, s \in H$ implies that w is horizontal. Thus (ii) holds.

To prove that the converse applies, it is obvious that $[p,w,q]_{\mathcal{A}_1}$ implies $[p,w,q]_{\mathcal{A}_3}$. Assume that $ii)$ holds, so $w = \mathrm{Label}(\mathbf{r})$ for $\mathbf{r} = t_1 \dots t_m \in \mathrm{Runs}(\mathcal{A}_2)$ with $\mathrm{start}(t_1) = r \in R$ and $\mathrm{end}(t_m) = s \in S$. By Lemma 3.12, as w is horizontal and r and s are in H, $\mathbf{r}$ must be direct run. Hence t_1 and t_m is not a matching parenthesizing transition pair. Thus it is possible to replace both $\langle$ and $\rangle$ with $\langle^{\mathrm{first}}$ and $\rangle^{\mathrm{last}}$, in the first and in the last transitions, if necessary. We can also substitute their closing and opening partners by $\rangle^{\mathrm{first}}$ and $\langle^{\mathrm{last}}$, if needed. Therefore the construction of $\mathcal{A}_3$ ensures that $[p,w,q]_{\mathcal{A}_3}$ holds. □

In the previous lemma we assumed that p and q were horizontal states, and R and S were sets of horizontal states. Of course an analogous statement can be formulated for vertical states and sets of vertical states as well.

3.6 Normal Forms

In this section we shall prove that every PA is equivalent to one in normal form, i.e. with a single initial and a single final state. If both of these states are horizontal then we will say that the PA is in horizontal normal form. If they are vertical states, then the normal form will be called vertical.

Definition 3.22 ([Ném06]) *We say that a PA is in* horizontal normal form *if it has a single initial state i_h, and a single final state f_h, and both i_h and f_h are horizontal states; moreover there is no transition into i_h or from f_h. Automata in* vertical normal form *can be defined in a similar way.*

Lemma 3.23 ([Ném06]) *For every PA $\mathcal{A}$, there exists an equivalent PA $\mathcal{A}_h$ in horizontal normal form and an equivalent PA $\mathcal{A}_v$ in vertical normal form.*

Proof. First we prove that for every PA $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ there exists a PA $\mathcal{A}^{\cap\mathcal{H}}$ in horizontal normal form that accepts exactly the horizontal biwords accepted by $\mathcal{A}$, that is,

$$L(\mathcal{A}^{\cap\mathcal{H}}) = L(\mathcal{A}) \cap \mathcal{H},$$

where $\mathcal{H}$ denotes the set of all horizontal biwords. For this let

$$T := \{\, (s,t) \mid \exists i \in I \cap V, \exists f \in F \cap V, \exists \langle, \rangle \in \Omega \;:\; (i, \langle, s),\ (t, \rangle, f) \in \gamma \,\},$$

and assume that $T = \{\, (s_1,t_1), (s_2,t_2), \ldots, (s_n,t_n) \,\}$. The elements of T can be called *pseudo initial-final pairs of horizontal states* as they are possibly not real initial and final states, but any run between them on a horizontal biword belongs to the accepted language. Moreover, $\mathcal{A}$ has a direct run on all horizontal biwords of $L(\mathcal{A})$, either between a real initial and final states, or between a pseudo initial-final pair of horizontal states.

Now let $\mathcal{A}_0$ be the automaton without transitions which has just two states, namely an initial horizontal state i_h and a final horizontal state f_h.

With the help of the substitution product, we define

$$\begin{aligned} \mathcal{A}_1 &:= \mathcal{A}_0 *_{[i_h \to I\cap H, F\cap H \to f_h]} \mathcal{A}, \\ \mathcal{A}_{k+1} &:= \mathcal{A}_k *_{[i_h \to s_k, t_k \to f_h]} \mathcal{A} \quad \text{for } k = 1, \ldots, n, \\ \mathcal{A}^{\cap\mathcal{H}} &:= \mathcal{A}_{n+1}. \end{aligned}$$

Using Lemma 3.21 and Proposition 3.16, it is straightforward to verify that $L(\mathcal{A}^{\cap\mathcal{H}}) = L(\mathcal{A}) \cap \mathcal{H}$, as claimed.

Similarly, there is an automaton $\mathcal{A}^{\cap\mathcal{V}}$ in vertical normal form which just accepts the vertical biwords accepted by $\mathcal{A}$. Let i_v and f_v denote the (single) initial and final vertical states of $\mathcal{A}^{\cap\mathcal{V}}$.

Next, we can construct $\mathcal{A}_h$ by taking the disjoint union of $\mathcal{A}^{\cap\mathcal{H}}$ and $\mathcal{A}^{\cap\mathcal{V}}$ and adding two new parenthesizing transitions, $(i_h, \{, i_v)$ and $(f_v, \}, f_h)$, where { and } is a new pair of parentheses. Naturally we no longer regard i_v and f_v as initial and final states. In order to accept the singleton biwords, we also define (i_h, σ, f_h) for each singleton biword $\sigma \in L(\mathcal{A})$. As $\mathcal{A}^{\cap\mathcal{H}}$ accepts all horizontal, and $\mathcal{A}^{\cap\mathcal{V}}$ all vertical biwords of $L(\mathcal{A})$, the resulting automaton is equivalent to $\mathcal{A}$. Moreover, $\mathcal{A}$ has a single initial state i_h and a single finite state f_h, as required. Again, $\mathcal{A}_v$ can be defined symmetrically. □

3.7 The ξ-substitution and Closure Properties

In this section we will consider some basic operations on binoid languages, and we will show that the class of regular languages is closed under them. First we will prove closure under the operation of ξ-substitution as the other closure properties can be easily derived from this.

Definition 3.24 *Let $L, L_1, L_2 \subseteq \Sigma^*(\bullet, \circ)$. We define the following operations, called* horizontal *(or* series product*),* vertical *(or* parallel product*),* horizontal iteration *(or* series iteration*) and* vertical iteration *(or* parallel iteration*).*

$$\begin{aligned} L_1 \bullet L_2 &:= \{w_1 \bullet w_2 \mid w_1 \in L_1,\ w_2 \in L_2\}, \\ L_1 \circ L_2 &:= \{w_1 \circ w_2 \mid w_1 \in L_1,\ w_2 \in L_2\}, \\ L^{*\bullet} &:= \{w_1 \bullet \ldots \bullet w_n \mid w_i \in L,\ i \in [n], \quad n \geq 0\}, \\ L^{*\circ} &:= \{w_1 \circ \ldots \circ w_n \mid w_i \in L,\ i \in [n], \quad n \geq 0\}. \end{aligned}$$

Furthermore, we let $L^{+\bullet} := L \bullet L^{*\bullet}$ and $L^{+\circ} := L \bullet L^{*\circ}$.

Now assume that $L_1 \subseteq (\Sigma \cup \{\, \xi \,\})^*(\bullet, \circ)$ and $L_2 \subseteq \Sigma^*(\bullet, \circ)$ the *ξ-substitution* of L_2 into L_1 will be denoted by $L_1[L_2/\xi]$. It is obtained by substituting non-uniformly biwords in L_2 for ξ in the members of L_1. Here non-uniformly means that different occurrences of ξ in a biword from L_1 may be replaced by different biwords from L_2. The formal definition is rather long and gives us no real insight, hence it will be omitted. It can be obtained in the pattern of tree languages. Please see Definition 4.3. and Definition 4.5. of [GS84] for details.

Theorem 3.25 ([ÉN04]) *The class* Reg *of regular binoid languages is (effectively) closed under ξ-substitution, i.e. if* $L_1 \subseteq (\Sigma \cup \{\,\xi\,\})^*(\bullet,\circ)$, $L_2 \subseteq \Sigma^*(\bullet,\circ)$, *then* $L_1, L_2 \in$ Reg *implies* $L_1[L_2/\xi] \in$ Reg.

Proof. Suppose that automaton $\mathcal{A}_1 = (S_1, H_1, V_1, \Sigma \cup \{\,\xi\,\}, \Omega_1, \delta_1, \gamma_1, I_1, F_1)$ accepts L_1. For the sake of simplicity, assume that there is only one single ξ-transition (p, ξ, q) in δ_1. If there were more ξ-transitions, they could be handled in the same way. Moreover, assume that p and q are horizontal states. The vertical case can be handled in a similar way, of course.

Now consider an automaton $\mathcal{A}_2$ that accepts L_2. Furthermore, assume that $\mathcal{A}_2$ is in horizontal normal form, hence $\mathcal{A}_2$ can be written as $\mathcal{A}_2 = (S_2, H_2, V_2, \Sigma, \Omega_2, \delta_2, \gamma_2, \{\,i_h\,\}, \{\,f_h\,\})$.

Recall that an accepting run is a run from an initial state to a final state. Our aim is to build an automaton $\mathcal{A}_3$ that extends $\mathcal{A}_1$ and is also capable of realizing accepting runs of $\mathcal{A}_2$ between p and q. At first sight it seems sufficient to add a disjoint copy of $\mathcal{A}_2$ to $\mathcal{A}_1$, and duplicate the transitions starting from i_h using p as the origin, and duplicate the transitions arriving at f_h using q as the target. In addition, the transitions (i_h, σ, i_f) should also be copied as (p, σ, q), where $\sigma \in \Sigma$.

However this is not enough as in this construction we might have superfluous runs whose label is not in $L_1[L_2/\xi]$. Indeed, if a run passes through the component $\mathcal{A}_2$ from p to q more than once, it may happen that during the first time the automaton opens some parentheses, then it returns to $\mathcal{A}_1$, and the parentheses are closed only in a later visit of $\mathcal{A}_2$ from p to q. Such runs may have no subruns between p and q, so their labels do not necessarily belong to $L_1[L_2/\xi]$. Hence, we must exclude the possibility of leaving behind opened parentheses during subruns from p to q in $\mathcal{A}_2$.

We can achieve this, for example, by using two copies of $\mathcal{A}_2$. Let us denote them by $\mathcal{A}_2'$ and $\mathcal{A}_2''$. Transition (p, ξ, q) is replaced by $\mathcal{A}_2'$ in a similar way as before. But whenever the first parenthesis is opened (either from $\mathcal{A}_2'$ or directly from p), we will mark this parenthesis with a superscript "first", and the automaton will enter into the second component $\mathcal{A}_2''$. Furthermore, to return back from $\mathcal{A}_2''$ to $\mathcal{A}_2'$ or q there is no other way but to close the marked parenthesis. Since the first parenthesis has to be closed, all other parentheses that were opened after leaving p have to be closed as well before the run reaches q.

For a formal description, let introduce some abbreviations:

$$H_2' := \{\, h' \mid h \in H_2 \,\}, \qquad H_2'' := \{\, h'' \mid h \in H_2 \,\},$$
$$V_2' := \{\, v' \mid v \in V_2 \,\}, \qquad V_2'' := \{\, v'' \mid v \in V_2 \,\},$$

$$S_2' := H_2' \cup V_2', \qquad S_2'' := H_2'' \cup V_2'',$$
$$\delta_2' := \{\, (s', \sigma, t') \mid (s, \sigma, t) \in \delta_2 \,\}, \qquad \delta_2'' := \{\, (s'', \sigma, t'') \mid (s, \sigma, t) \in \delta_2 \,\},$$
$$\gamma_2' := \{\, (s', \omega, t') \mid (s, \omega, t) \in \gamma_2 \,\}, \qquad \gamma_2'' := \{\, (s'', \omega, t'') \mid (s, \omega, t) \in \gamma_2 \,\}.$$

Next, take

$$\mathcal{A}_3 := (S_3, H_3, V_3, \Sigma, \Omega_3, \delta_3, \gamma_3, I_1, F_1),$$

where

$$\begin{aligned}
S_3 &:= S_1 \cup S_2' \cup S_2'', \quad H_3 := H_1 \cup H_2' \cup H_2'', \quad V_3 := V_1 \cup V_2' \cup V_2'', \\
\Omega_3 &:= \Omega_1 \cup \Omega_2 \cup \{\, \langle^{\text{first}}, \rangle^{\text{first}} \mid \langle, \rangle \in \Omega_2 \,\}, \\
\delta_3 &:= \delta_1 \setminus \{\, (p, \xi, q) \,\} \cup \delta_2' \cup \delta_2'' \\
&\quad \cup \{\, (p, a, x') \mid a \in \Sigma, x \in S_2, (i_h, a, x) \in \delta_2 \,\} \\
&\quad \cup \{\, (y', b, q) \mid y \in S_2, b \in \Sigma, (y, b, f_h) \in \delta_2 \,\} \\
&\quad \cup \{\, (p, \sigma, q) \mid \sigma \in \Sigma, (i_h, \sigma, f_h) \in \delta_2 \,\}, \\
\gamma_3 &:= \gamma_1 \cup \gamma_2'' \\
&\quad \cup \{\, (p, \langle^{\text{first}}, w'') \mid \langle \in \Omega_{op}, w \in S_2, (i_h, \langle, w) \in \gamma_2 \,\} \\
&\quad \cup \{\, (z'', \rangle^{\text{first}}, q) \mid z \in S_2, \rangle \in \Omega_{cl}, (z, \rangle, f_h) \in \gamma_2 \,\} \\
&\quad \cup \{\, (r', \langle^{\text{first}}, s'') \mid r \in S_2, \langle \in \Omega_{op}, s \in S_2, (r, \langle, s) \in \gamma_2 \,\} \\
&\quad \cup \{\, (t'', \rangle^{\text{first}}, u') \mid t \in S_2, \rangle \in \Omega_{cl}, u \in S_2, (t, \rangle, u) \in \gamma_2 \,\}.
\end{aligned}$$

Our intention here is to prove the equation $L(\mathcal{A}_3) = L_1[L_2/\xi]$. The inclusion $L_1[L_2/\xi] \subseteq L(\mathcal{A}_3)$ directly follows from the construction: in all runs of $\mathcal{A}_1$ any accepting run of $\mathcal{A}_2$ can be substituted in the places of the transition (p, ξ, q) .

As for the converse inclusion, the construction ensures the following statement. Assume that $\mathbf{t}$ is a sequence of transitions from p to q without any transition in $\mathcal{A}_1$, moreover let $\mathbf{t}$ reach p and q only once (at the beginning and at the end). Then it follows from the definition of $\mathcal{A}_3$ that in $\mathbf{t}$ every parenthesis that is opened will also be closed, hence if $\mathbf{t}$ is a subsequence of a run of $\mathcal{A}_3$, then it is also a subrun of $\mathcal{A}_3$ from p to q. Therefore there is a biword w' such that $\mathrm{Label}(\mathbf{t}) = w'$. Again, by the construction $[i_h, w', f_h]_{\mathcal{A}_2}$, so $w' \in L_2$ holds.

Now assume that $w \in L(\mathcal{A}_3)$. This means that there is an accepting run $\mathbf{r}$ of $\mathcal{A}_3$ such that $\mathrm{Label}(\mathbf{r}) = w$. Let $\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_n}$ denote those subsequences of $\mathbf{r}$ that start from p, end at q, are without transitions of $\mathcal{A}_1$ and contain p and q only once. By the argument above, there are biwords $w_1, \ldots, w_n \in L_2$ such that $\mathrm{Label}(\mathbf{t_i}) = w_i$, for all $i \in [n]$. Now if in $\mathbf{r}$ we replace each $\mathbf{t_i}$ with a (p, ξ, q) transition, then we get a run

$\mathbf{r}'$ that contains transitions only from $\mathcal{A}_1$. Obviously, if we set $w' := \mathrm{Label}(\mathbf{r}')$, then $w' \in L_1$ holds. Moreover, if we substitute the occurrences of ξ in w' with $w_1, \ldots, w_n$ in turn, we will get w. Hence $w \in L_1[L_2/\xi]$. Therefore, $L(\mathcal{A}_3) = L_1[L_2/\xi]$, thus $L_1[L_2/\xi]$ is regular, as claimed. □

Using Theorem 3.25, we can immediately derive some further closure properties of Reg, like it was done in [LW00] for regular sp-poset languages.

Corollary 3.26 ([ÉN04]) *The class* Reg *of regular binoid languages is (effectively) closed under horizontal and vertical products, horizontal and vertical iterations, and homomorphisms.*

Proof. The statements immediately follow from Theorem 3.25. Indeed, if $\xi_1, \xi_2 \notin \Sigma$ then since $\{\xi_1 \bullet \xi_2\}$ and $\xi_1^{*\bullet}$ are regular,

$$\begin{aligned} L_1 \bullet L_2 &= (\{\xi_1 \bullet \xi_2\}[L_1/\xi_1])\,[L_2/\xi_2], \text{ and} \\ L_1^{*\bullet} &= \xi_1^{*\bullet}[L_1/\xi_1] \end{aligned}$$

are also regular. Evidently, the same holds for the vertical product and for the vertical iteration operation.

As homomorphisms can be expressed by sequences of substitutions, closure under homomorphisms also follows from closure under substitution. □

Later, after we prove the equality of the classes of regular and recognizable binoid languages, we will easily derive additional closure properties from the definition of recognizability (cf. Corollary 3.39).

3.8 Hierarchy theorems

In this session we will show that it is essential that in the definition of PA an arbitrary number of parentheses can be used. We will present two hierarchy results stated in Theorems 3.32 and 3.33. Let Reg_m denote the regular binoid languages that can be accepted by a PA with at most $m \geq 0$ pairs of parentheses. We will show that $\mathsf{Reg}_m \subsetneq \mathsf{Reg}_{m+1}$ for all $m \geq 0$. Hence the classes Reg_m form a strict hierarchy. Moreover, if $\mathsf{Reg}_m(\Sigma)$ is the set of all regular binoid languages over any fixed alphabet Σ that can be accepted by an automaton with at most m pairs of parentheses, then we also have $\mathsf{Reg}_m(\Sigma) \subsetneq \mathsf{Reg}_{m+1}(\Sigma)$, for all $m \geq 0$.

Our first goal is to define a language $\widetilde{L}$ that separates the classes Reg_2 and Reg_1, i.e. $\widetilde{L}$ is in $\mathsf{Reg}_2 \setminus \mathsf{Reg}_1$.

Definition 3.27 ([Ném04]) *Let Σ_2 denote the two-letter alphabet $\Sigma_2 = \{a, b\}$, and let*

$$\begin{aligned} \widetilde{L} &= \bigcup_{i=1}^{\infty} \widetilde{L}_{2i}, \text{ where} \\ \widetilde{L}_2 &= \{\, \sigma \bullet \langle \sigma' \circ \sigma' \rangle \bullet \sigma \mid \sigma, \sigma' \in \Sigma_2 \,\}, \\ \widetilde{L}_{2i+2} &= \{\, \sigma \bullet \langle \sigma' \circ w \circ \sigma' \rangle \bullet \sigma \mid \sigma, \sigma' \in \Sigma_2 \text{ and } w \in \widetilde{L}_{2i} \,\}, \text{ for all } i \geq 1. \end{aligned}$$

Definition 3.28 ([Ném04]) *For any words $u = \sigma_1\sigma_2 \ldots \sigma_n$ and $v = \rho_1\rho_2 \ldots \rho_n$ of even length over some alphabet Σ, let $P_{uv^{-1}}$ denote the biword whose term representation is*

$$P_{uv^{-1}} = \sigma_1 \bullet \langle \sigma_2 \circ \langle \sigma_3 \bullet \langle \sigma_4 \circ \ldots \langle \sigma_{n-1} \bullet \langle \sigma_n \circ \rho_n \rangle \bullet \rho_{n-1} \rangle \ldots \circ \rho_4 \rangle \bullet \rho_3 \rangle \circ \rho_2 \rangle \bullet \rho_1.$$

Note that $\widetilde{L} \subseteq \Sigma_2^+(\bullet, \circ)$ and $\widetilde{L} = \{\, P_{ww^{-1}} \mid w \in \Sigma_2^+,\ w \text{ has even length} \,\}$.

Lemma 3.29 ([Ném04]) *$\widetilde{L}$ can be accepted by a PA that has two pairs of parentheses, i.e. $\widetilde{L} \in \mathsf{Reg}_2(\Sigma_2)$.*

Proof. It is not hard to see that the automaton $\mathcal{A}_2$ in Figure 3.7 accepts $\widetilde{L}$. □

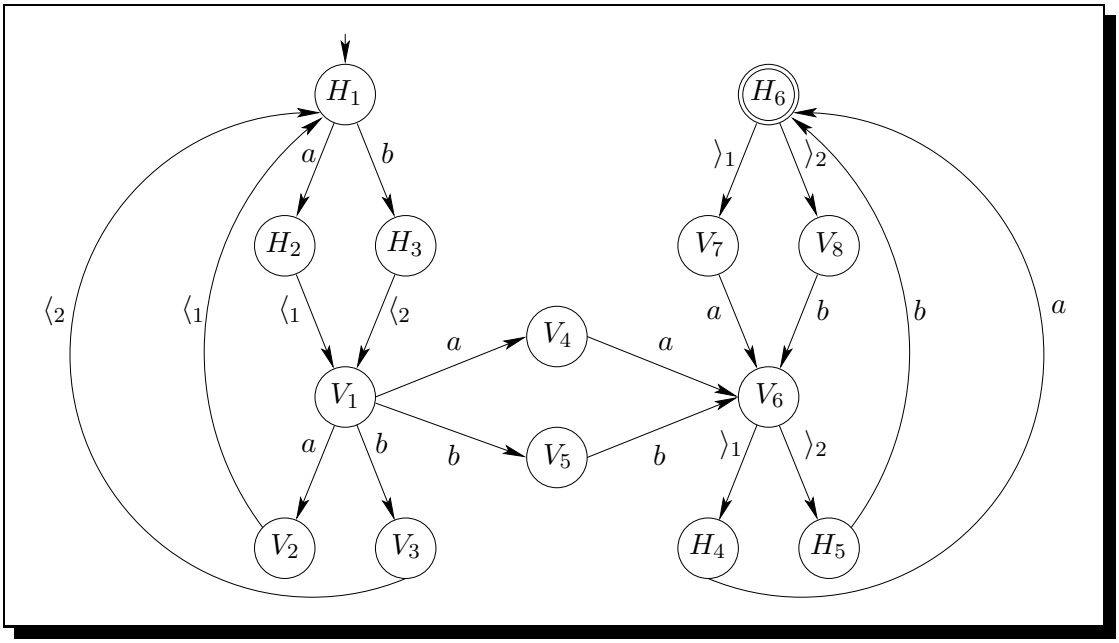

Figure 3.7: *An automaton $\mathcal{A}_2$ accepting $\widetilde{L}$.*

Our next goal is to show that $\widetilde{L}$ is not in $\mathsf{Reg}_1(\Sigma_2)$, but we need to make some further definitions before we can begin the proof.

Definition 3.30 ([Ném04]) *A* generalized state *of a PA $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ is a pair (q, α), where $q \in S$ and $\alpha \in \Omega_{op}^*$.*

Recall that the set of configuration of $\mathcal{A}$ introduced in Definition 3.11 is

$$\mathrm{Conf}(\mathcal{A}) = S \times \Omega_{op}^* \times \Big(\Sigma \cup \{\langle, \rangle\}\Big)^*,$$

so a generalized state is just the first two components of a configuration.

Now it is convenient to say that an automaton $\mathcal{A}$ *has a transition* from a generalized state (q_1, α_1) to a generalized state (q_2, α_2) with respect to the symbol $x \in \Sigma \cup \{\langle, \rangle\}$, if $(q_1, \alpha_1, x) \vdash (q_2, \alpha_2, \varepsilon)$ hold for the configurations. The notation for this will be $(q_1, \alpha_1) \overset{x}{\vdash} (q_2, \alpha_2)$.

An immediate consequence of this definition is that every run $[p, w, q]_{\mathcal{A}}$ on a nonempty and nonsingleton biword w corresponds to a sequence of transitions among the generalized states

$$(q_0, \varepsilon) \overset{x_1}{\vdash} (q_1, \alpha_1) \overset{x_2}{\vdash} (q_2, \alpha_2) \overset{x_3}{\vdash} \ldots \overset{x_n}{\vdash} (q_n, \varepsilon),$$

where $q_0 = p, q_n = q$ and $(q_i, x_i, q_{i+1}) \in \delta \cup \gamma$ for $i \in [n]$, and α_i is the sequence of opened but not yet closed parentheses after the first i steps of the run. Moreover, by Lemma 3.12, $x_1 \ldots x_n = w^{ctm-}$ if the run is direct, and $x_1 \ldots x_n = \langle w^{ctm-} \rangle$ if the run is indirect.

Lemma 3.31 ([Ném04]) *There is no PA with a single pair of parentheses which accepts $\widetilde{L}$, i.e. $\widetilde{L} \notin \mathsf{Reg}_1(\Sigma_2)$.*

Proof. Assuming the contrary is true, suppose that there is a PA $\mathcal{A} = (S, H, V, \Sigma_2, \Omega_1, \gamma, \delta, I, F)$ which accepts $\widetilde{L}$ with $\Omega_1 = \{\langle, \rangle\}$. Let n be an even integer greater than $|V|$. The number of all biwords $P_{ww^{-1}} \in \widetilde{L}$, where w is of length n, is 2^n. Each $P_{w_j w_j^{-1}}$ is accepted either via a direct run between two horizontal states, or via an indirect run between two vertical states. Thus, since $n - 1 + n - 1 = 2n - 2 < 2^n$ holds for all n, either there are n biwords accepted between horizontal states, or else there are n biwords accepted between vertical states. For simplicity we will assume it is the former case. This is not a real restriction as in the other case our proof would be essentially the same, only an opening and a closing parenthesizing transition needs to be added before and after the runs.

Thus let us take n distinct words of n letters $w_j \in \Sigma_2^n$, $j \in [n]$, and consider the biwords

$$P_{w_j w_j^{-1}} = \sigma_1^j \bullet \langle \sigma_2^j \circ \langle \sigma_3^j \bullet \langle \sigma_4^j \circ \ldots \langle \sigma_{n-1}^j \bullet \langle \sigma_n^j \circ \sigma_n^j \rangle \bullet \sigma_{n-1}^j \rangle \ldots \circ \sigma_4^j \rangle \bullet \sigma_3^j \rangle \circ \sigma_2^j \rangle \bullet \sigma_1^j,$$

where $\sigma_1^j \sigma_2^j \ldots \sigma_n^j = w_j$. Now, each $P_{w_j w_j^{-1}}$ is, by definition, in $\widetilde{L}$ but as we shall see, $\mathcal{A}$ must accept biwords that do not belong to $\widetilde{L}$.

Indeed, each accepting direct run of $\mathcal{A}$ on any $P_{w_j w_j^{-1}}$ must have the form

$$(q_0,\varepsilon) \overset{\sigma_1^j}{\vdash} (q_1,\varepsilon) \overset{\langle}{\vdash} (q_2,\langle\rangle) \overset{\sigma_2^j}{\vdash} (q_3,\langle\rangle) \overset{\langle}{\vdash} (q_4,\langle\langle\rangle) \overset{\sigma_3^j}{\vdash} \dots \overset{\sigma_{n-1}^j}{\vdash} (q_{2n-3},\langle^{n-2}) \overset{\langle}{\vdash} (q_{2n-2},\langle^{n-1}) \overset{\sigma_n^j}{\vdash}$$
$$(q_{2n-1},\langle^{n-1}) \overset{\sigma_n^j}{\vdash} (q_{2n},\langle^{n-1}) \overset{\rangle}{\vdash} (q_{2n+1},\langle^{n-2}) \overset{\sigma_{n-1}^j}{\vdash} (q_{2n+2},\langle^{n-2}) \overset{\rangle}{\vdash} \dots \overset{\sigma_2^j}{\vdash} (q_{4n-4},\langle\rangle) \overset{\rangle}{\vdash}$$
$$(q_{4n-3},\varepsilon) \overset{\sigma_1^j}{\vdash} (q_{4n-2},\varepsilon), \text{where } q_0 \in I \cap H \text{ and } q_{4n-2} \in F \cap H.$$

For our investigation the main point is that after processing the "first half" of $P_{w_j w_j^{-1}}$, (i.e. after the first $2n-1$ transitions) the automaton enters a generalized state $(q_{2n-1},\langle^{n-1})$, where q_{2n-1} is a vertical state and after an additional $2n-1$ transitions, the run ends in (q_{4n-2},ε).

Given $P_{w_j w_j^{-1}}$, let i_j, v_j and f_j denote the states q_0, q_{2n-1}, and q_{4n-2}, respectively, which appear in the above accepting run of $\mathcal{A}$ on the biword $P_{w_j w_j^{-1}}$. Moreover, let us abbreviate the transition sequences determined by the first $2n-1$ and the second $2n-1$ transitions by

$$(i_j,\varepsilon) \overset{P_{w_j *}}{\vdash} (v_j,\langle^{n-1}) \text{ and } (v_j,\langle^{n-1}) \overset{P_{*w_j^{-1}}}{\vdash} (f_j,\varepsilon),$$

respectively.

Now we have n vertical states $v_1, v_2, \dots, v_n$, but we have chosen $n > |V|$, so there must be two indices $k \neq l$ such that $v_k = v_l$. Hence the transition sequence

$$(i_k,\varepsilon) \overset{P_{w_k *}}{\vdash} (v_k,\langle^{n-1}) = (v_l,\langle^{n-1}) \overset{P_{*w_l^{-1}}}{\vdash} (f_l,\varepsilon)$$

corresponds to a valid run of $\mathcal{A}$, showing that $P_{w_k w_l^{-1}}$ is accepted by $\mathcal{A}$. But by definition $P_{w_k w_l^{-1}} \notin \widetilde{L}$, which is a contradiction. Thus no $\mathcal{A}$ with a single pair of parentheses can accept $\widetilde{L}$, so $\widetilde{L} \notin \mathsf{Reg}_1(\Sigma_2)$. □

The previous theorem can be extended to any $m \geq 1$ as follows.

Theorem 3.32 ([Ném04]) *For all $m \geq 1$ there exists a language $\widetilde{L}(\Sigma_m)$ that can be accepted by an automaton with m but not with $m-1$ pairs of parentheses. Thus the classes $\mathsf{Reg}_0 \subsetneq \mathsf{Reg}_1 \subsetneq \mathsf{Reg}_2 \subsetneq \dots$ form a strict hierarchy of regular (i.e. recognizable) binoid languages.*

Proof. This is trivial for the case $m = 1$, and $\mathsf{Reg}_1 \subsetneq \mathsf{Reg}_2$ was shown in Lemma 3.29 and Lemma 3.31. As for $m \geq 3$, we will show how the proofs of these two lemmas can be generalized. Let

$$\widetilde{L}(\Sigma_m) = \{\, P_{ww^{-1}} \mid w \in \Sigma_m^*,\ w \text{ has even length}\}.$$

Note that $\widetilde{L} = \widetilde{L}(\Sigma_2)$. In Lemma 3.29, the automaton $\mathcal{A}_2$ accepting $\widetilde{L}$ can easily be generalized to an automaton $\mathcal{A}_m$ accepting $\widetilde{L}(\Sigma_m)$ by using m different parentheses corresponding to the m letters.

In order to see that $\widetilde{L}(\Sigma_m) \notin \mathsf{Reg}_{m-1}$, suppose on the contrary that $\widetilde{L}(\Sigma_m) = L(\mathcal{A})$ for some automaton $\mathcal{A} = (S, H, V, \Sigma_m, \Omega_{m-1}, \gamma, \delta, I, F)$, where we have $\Omega_{m-1} = \{ \langle_1, \rangle_1, \langle_2, \rangle_2, \ldots, \langle_{m-1}, \rangle_{m-1} \}$. Now we choose pairwise different words $w_1, w_2, \ldots, w_n$ in Σ_m^n as before, but this time we give the value of n later.

We notice that after reading the first half of $P_{w_j w_j^{-1}}$, where $P_{w_j w_j^{-1}}$ is defined as in the proof of Lemma 3.31, $\mathcal{A}$ is necessarily in a generalized state of the form

$$(v_j, \langle_{i_1} \langle_{i_2} \ldots \langle_{i_{n-1}}), \qquad (*)$$

where $v_j \in V$ and $\langle_{i_k} \in \Omega_{m-1}$ for all $k \in [n-1]$.

But the number of this type of generalized states, $|V| \cdot (m-1)^{n-1}$ is asymptotically smaller than m^n, the number of words w_j of length n, which is the number of all biwords of the form $P_{w_j w_j^{-1}}$. Thus, in the same way as above, n can always be chosen such that $\mathcal{A}$ accepts at least one biword $P_{w_j w_k^{-1}}$ for some $w_j \neq w_k$. □

In the proof of Theorem 3.32 we used the m-letter alphabet Σ_m to show that $\mathsf{Reg}_{m-1}(\Sigma_m) \subsetneq \mathsf{Reg}_m(\Sigma_m)$. However this proper inclusion holds for every Σ.

Theorem 3.33 ([Ném04]) *For each alphabet Σ the classes $\mathsf{Reg}_0(\Sigma) \subsetneq \mathsf{Reg}_1(\Sigma) \subsetneq \mathsf{Reg}_2(\Sigma) \ldots$ form a strict hierarchy in $\mathsf{Reg}(\Sigma)$.*

Proof. It is sufficient to prove this claim for a one-letter alphabet. So assume that $\Sigma = \Sigma_1 = \{a\}$ and $\Sigma_m = \{ a_1, a_2, \ldots, a_m \}$. Furthermore, suppose that $m \geq 1$. Let h denote the binoid homomorphism $\Sigma_m^*(\bullet, \circ) \to \Sigma_1^*(\bullet, \circ)$ determined by the assignment

$$a_i \mapsto \underbrace{a \bullet a \bullet \ldots \bullet a}_{i \text{ times}}, \text{for all } a_i \in \Sigma_m.$$

Now we claim that the language $h(\widetilde{L}(\Sigma_m))$ is in $\mathsf{Reg}_m(\Sigma_1) \setminus \mathsf{Reg}_{m-1}(\Sigma_1)$.

Indeed, it is not hard to modify the automaton $\mathcal{A}_m$ in the proof of Theorem 3.32 to accept $h(\widetilde{L}(\Sigma_m))$ instead of $\widetilde{L}(\Sigma_m)$. On the other hand, after reading the "first half" of a biword $h(P_{w_j w_j^{-1}})$, any PA with $m-1$ pairs of parentheses must be in a generalized state of $(*)$ as before. Thus the same cardinality argument can be applied to show that $h(\widetilde{L}(\Sigma_m))$ is not in $\mathsf{Reg}_{m-1}(\Sigma_1)$. □

3.9 Recognizability

The concept of recognizability is of an unquestionable importance in theoretical computer science [Wei04a]. It is so, because it leads to decision algorithms, the class of recognizable languages has nice closure properties, and according to the variety theorem of Eilenberg [Eil76, Pin86] (which has various generalizations beyond the classical case of words) important language classes can be characterized by the algebraic structures of their recognizing finite algebras.

Furthermore, recall that in the case of binoid languages, while we have some freedom in defining the concept of automata and regular languages, the concept of recognizable languages is fixed: it is dictated by the fact that the we investigate objects where two associative operations are present. Therefore the algebraic framework of binoid languages is necessarily the theory of binoids. Soon in Theorem 3.35 we will prove the equivalence of regularity and recognizability. This will justify the point that our concept of PA captures the essential and robust class of recognizable binoid languages. As was mentioned earlier, this result will lead us to some further closure properties of Reg. Moreover, in Section 3.10.3 on the basis of this equivalence, we can derive some decision procedures relating to various properties of regular binoid languages.

First we start with the concept of recognizable binoid languages. Recall that a binoid is an algebra $B = (B, \bullet, \circ, 1)$ where $\bullet$ and $\circ$ are associative operations on the set B, and 1 is the identity for both operations. Homomorphisms and congruences of bisemigroups are defined as usual. A congruence, or equivalence relation of a bisemigroup is of *finite index* if the partition induced by the relation has a finite number of blocks.

Definition 3.34 *A binoid language $L \subseteq \Sigma^*(\bullet, \circ)$ is* recognizable *if there is a finite binoid B, a homomorphism $h : \Sigma^*(\bullet, \circ) \to B$, and a set $F \subseteq B$ with $L = h^{-1}(F)$.*

Theorem 3.35 ([ÉN04]) $\mathsf{Rec} = \mathsf{Reg}$*, i.e. a binoid language $L \subseteq \Sigma^*(\bullet, \circ)$ is recognizable iff L is regular.*

Proof. In our proof we show how to construct a finite binoid and a homomorphism from a parenthesizing automaton, and conversely, how to construct a PA from a finite binoid and a homomorphism recognizing a given binoid language L.

Let $L \subseteq \Sigma^*(\bullet, \circ)$ be a binoid language, and let $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ be a PA accepting L. Define the relation $\sim$ on $\Sigma^*(\bullet, \circ)$ as follows:

$$x \sim y \quad \Leftrightarrow \quad (\forall\, p, q \in S :\ [p, x, q]_{\mathcal{A}} \Leftrightarrow [p, y, q]_{\mathcal{A}}).$$

It is clear that $\sim$ is an equivalence relation on $\Sigma^*(\bullet,\circ)$ and that it has index at most $2^{m^2+n^2}$, where m and n denote the number of horizontal and vertical states of $\mathcal{A}$, respectively. We shall verify that $\sim$ is a congruence relation that saturates L. Thus the finite binoid $\Sigma^*(\bullet,\circ)/\sim$ recognizes L. Next, suppose that $x \sim y$, and let z be an arbitrary element of $\Sigma^*(\bullet,\circ)$. We need to show that $x \bullet z \sim y \bullet z$, $z \bullet x \sim z \bullet y$, $x \circ z \sim y \circ z$ and that $z \circ x \sim z \circ y$. The argument is based on the following lemma, which is a straightforward consequence of Definition 3.3.

Lemma 3.36 ([ÉN04]) *The following hold for any biwords x and z in $\Sigma^*(\bullet,\circ)$.*

(i) *If $p,q \in H$, then $[p, x \bullet z, q]_{\mathcal{A}} \Leftrightarrow \exists r \in H :\ [p,x,r]_{\mathcal{A}}$ and $[r,z,q]_{\mathcal{A}}$.*

(ii) *If $p,q \in V$, then $[p, x \bullet z, q]_{\mathcal{A}} \Leftrightarrow \exists \langle_k, \rangle_k \in \Omega,\ p', r', q' \in H :$*
$(p, \langle_k, p') \in \gamma,\ [p', x, r']_{\mathcal{A}},\ [r', z, q']_{\mathcal{A}},$ and $(q', \rangle_k, q) \in \gamma$.

Similar statements hold for the vertical product.

We will now show that $x \bullet z \sim y \bullet z$. First, let $p,q \in H$, then by Lemma 3.36,

$$\begin{aligned} [p, x \bullet z, q]_{\mathcal{A}} &\Leftrightarrow \exists r : [p,x,r]_{\mathcal{A}} \text{ and } [r,z,q]_{\mathcal{A}} \\ &\Leftrightarrow \exists r : [p,y,r]_{\mathcal{A}} \text{ and } [r,z,q]_{\mathcal{A}} \\ &\Leftrightarrow [p, y \bullet z, q]_{\mathcal{A}}\ . \end{aligned}$$

In the other case $p,q \in V$, and

$$\begin{aligned} &[p, x \bullet z, q]_{\mathcal{A}} \\ &\Leftrightarrow \exists \langle_k, \rangle_k \in \Omega, p', r', q' \in H : (p, \langle_k, p') \in \gamma, [p', x, r']_{\mathcal{A}}, [r', z, q']_{\mathcal{A}}, (q', \rangle_k, q) \in \gamma \\ &\Leftrightarrow \exists \langle_k, \rangle_k \in \Omega, p', r', q' \in H : (p, \langle_k, p') \in \gamma, [p', y, r']_{\mathcal{A}}, [r', z, q']_{\mathcal{A}}, (q', \rangle_k, q) \in \gamma \\ &\Leftrightarrow [p, y \bullet z, q]_{\mathcal{A}}\ . \end{aligned}$$

So $x \sim y$ implies $x \bullet z \sim y \bullet z$. One can verify in a similar way that $x \sim y$ also implies $z \bullet x \sim z \bullet y$, $x \circ z \sim y \circ z$ and $z \circ x \sim z \circ y$. It is also obvious that $\varepsilon \bullet x \sim x \bullet \varepsilon \sim \varepsilon \circ x \sim x \circ \varepsilon \sim x$ for all $x \in \Sigma^*(\bullet,\circ)$. Thus $\sim$ is a congruence relation on the binoid $\Sigma^*(\bullet,\circ)$.

Finally, $\sim$ saturates L, since $x \in L$ and $x \sim y$ imply $y \in L$:

$$\begin{aligned} x \in L &\Rightarrow \exists i \in I,\ \exists f \in F :\ [i,x,f]_{\mathcal{A}} \\ &\Rightarrow \exists i \in I,\ \exists f \in F :\ [i,y,f]_{\mathcal{A}} \\ &\Rightarrow y \in L. \end{aligned}$$

Hence $\mathsf{Reg} \subseteq \mathsf{Rec}$.

As for the inclusion $\mathsf{Rec} \subseteq \mathsf{Reg}$, let $B = (B, \bullet, \circ, 1)$ be a finite binoid, $\varphi : \Sigma^*(\bullet, \circ) \to B$ a homomorphism, and $F \subseteq B$, and let $L = \varphi^{-1}(F)$. In order to prove that L is regular we shall construct a PA from B, φ and F:

$$\mathcal{A}(B, \varphi, F) \quad := \quad (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F), \text{ where}$$

- $S = H \cup V$, where $H = \{(s, H) \mid s \in B\}$, $V = \{(s, V) \mid s \in B\}$,
- $\Omega = \{\langle_s, \rangle_s \mid s \in B\}$,
- $I = \{(1, H)\}$,
- $F = \{(f, H) \mid f \in F\}$.

Below we will use the sort notation s^H and s^V for (s, H) and (s, V), respectively. We define δ and γ by using the operations of B. For all $s, t \in B$ and $a \in \Sigma$, let

$$(s^H, a, t^H) \in \delta \text{ iff } s \bullet \varphi(a) = t, \text{ and} \tag{3.1}$$
$$(s^V, a, t^V) \in \delta \text{ iff } s \circ \varphi(a) = t. \tag{3.2}$$

Moreover, let $(s^H, \langle_s, 1^V) \in \gamma$, for all $s \in B$, and $(u^V, \rangle_s, t^H) \in \gamma$, for all $s, u, t \in B$ with $s \bullet u = t$. In much the same way for the vertical product, let $(s^V, \langle_s, 1^H) \in \gamma$, for all $s \in B$, and $(u^H, \rangle_s, t^V) \in \gamma$, for all $s, u, t \in S$ such that $s \circ u = t$.

Example 3.37 As an exception, in this example biwords will be denoted by capital letters. The construction of $\mathcal{A}(B, \varphi, F)$ is illustrated in Figure 3.8. First suppose that $s \bullet u = t$ and $s \bullet r = w$ in B, then we have the following parenthesizing transitions in $\mathcal{A}(B, \varphi, F)$:

$$(s^H, \langle_s, 1^V),\ (u^V, \rangle_s, t^H),\ (r^V, \rangle_s, w^H) \in \gamma.$$

If P, Q_1, Q_2, R, Q' and R' are biwords such that $\varphi(P) = s$, $\varphi(Q_1) = v$, $v \circ \varphi(Q_2) = u$ and $t \bullet \varphi(R) = f$, then the existence of a run $[1^H, P \bullet \langle Q_1 \circ Q_2 \rangle \bullet R, f^H]$ can be inferred from the runs $[1^H, P, s^H]$, $[1^V, Q_1, v^V]$, $[v^V, Q_2, u^V]$, $[t^H, R, f^H]$:

$$\begin{array}{llll}
[1^V, Q_1, v^V]_{\mathcal{A}}, & [v^V, Q_2, u^V]_{\mathcal{A}}, & & \text{hence } [1^V, Q_1 \circ Q_2, u^V]_{\mathcal{A}}\,; \\
(s^H, \langle_s, 1^V) \in \gamma, & [1^V, Q_1 \circ Q_2, u^V]_{\mathcal{A}}, & (u^V, \rangle_s, t^H) \in \gamma, & \text{hence } [s^H, Q_1 \circ Q_2, t^H]_{\mathcal{A}}\,; \\
[1^H, P, s^H]_{\mathcal{A}}, & [s^H, Q_1 \circ Q_2, t^H]_{\mathcal{A}}, & [t^H, R, f^H]_{\mathcal{A}}, & \text{hence } [1^H, P \bullet \langle Q_1 \circ Q_2 \rangle \bullet R, f^H]_{\mathcal{A}}\,.
\end{array}$$

Similarly, if $\varphi(Q') = r$ and $w \bullet \varphi(R') = z$, and if $[1^H, P, s^H]$, $[1^V, Q', r^V]$, and $[w^H, R', z^H]$, then we have $[1^H, P \bullet Q' \bullet R', z^H]_{\mathcal{A}}$.

In order to prove $\mathsf{Rec} \subseteq \mathsf{Reg}$ it is enough to show the following:

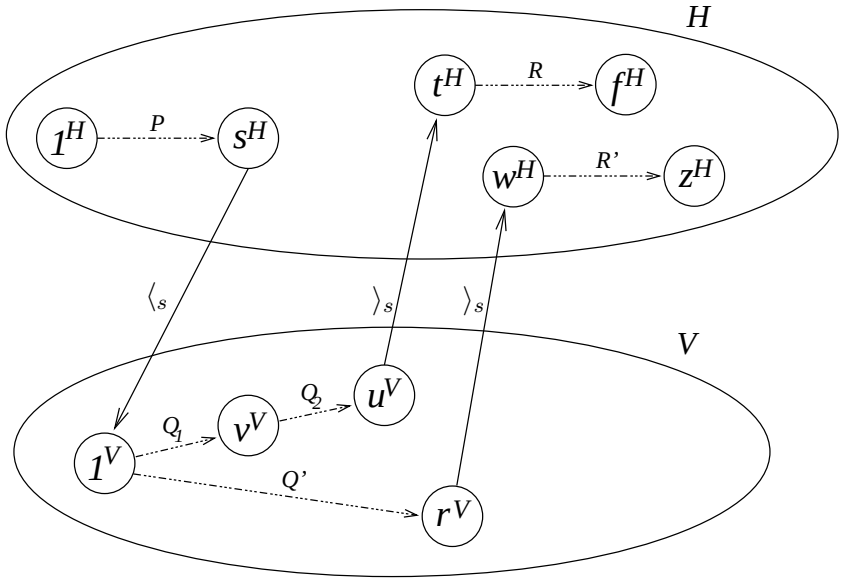

Figure 3.8: *The construction of* $\mathcal{A}(B, \varphi, F)$.

Lemma 3.38 *With the notations used above, for all* $w \in \Sigma^*(\bullet, \circ)$ *and* $s, t \in B$,

$$\begin{aligned} [s^H, w, t^H]_{\mathcal{A}} &\Leftrightarrow s \bullet \varphi(w) = t; \\ [s^V, w, t^V]_{\mathcal{A}} &\Leftrightarrow s \circ \varphi(w) = t. \end{aligned}$$

Proof of Lemma 3.38. We will apply induction on the construction of w. According to Definition 3.3, there are six cases. These are:
(E) If $w = \varepsilon$, then $s = t$ and $\varphi(w) = 1$, hence $s \bullet \varphi(w) = t$ and $s \circ \varphi(w) = t$ are obvious.
(S) If $w = a \in \Sigma$, then our statements hold by the definition of δ, i.e. by (3.1) and (3.2).
(HH) If $s^H, t^H \in H$ and w has a maximal horizontal decomposition $w = w_1 \bullet w_2 \bullet \ldots \bullet w_n$, $n \geq 2$, then

$$\begin{aligned} [s^H, w, t^H]_{\mathcal{A}} &\Leftrightarrow \exists r_1^H, r_2^H, \ldots, r_{n-1}^H, r_0^H = s^H, r_n^H = t^H : [r_{i-1}^H, w_i, r_i^H]_{\mathcal{A}},\ i \in [n] \\ &\Leftrightarrow \exists r_1, r_2, \ldots, r_{n-1}, r_0 = s, r_n = t : r_i = r_{i-1} \bullet \varphi(w_i),\ i \in [n] \\ &\Leftrightarrow s \bullet \varphi(w_1) \bullet \varphi(w_2) \bullet \ldots \bullet \varphi(w_n) = t \\ &\Leftrightarrow s \bullet \varphi(w) = t. \end{aligned}$$

The second equivalence above follows from the induction hypothesis.
(VV) If $s^V, t^V \in V$ and w has a maximal vertical decomposition $w = w_1 \circ w_2 \circ \ldots \circ w_n$, $n \geq 2$, then the proof is analogous to (HH).

(HV) If $s^H, t^H \in H$ and w has a maximal vertical decomposition $w = w_1 \circ w_2 \circ \ldots \circ w_n$, $n \geq 2$, then

$$\begin{aligned}
[s^H, w, t^H]_{\mathcal{A}} &\Leftrightarrow \exists u^V \in V : \ (s^H, \langle_s, 1^V) \in \gamma, \ [1^V, w, u^V]_{\mathcal{A}}, \ (u^V, \rangle_s, t^H) \in \gamma \\
&\Leftrightarrow \exists u \in B : \ s \bullet u = t \text{ and } 1 \circ \varphi(w) = u \\
&\Leftrightarrow s \bullet \varphi(w) = t.
\end{aligned}$$

The second equivalence above follows from the definition of γ and from case (VV).
(VH) If $s^V, t^V \in V$ and w has a maximal horizontal decomposition $w = w_1 \bullet w_2 \bullet \ldots \bullet w_n$, $n \geq 2$, then the proof is analogous to (HV). This concludes the proof of Lemma 3.38. □

Proof of Theorem 3.35, completed. For all $w \in \Sigma^*(\bullet, \circ)$

$$\begin{aligned}
\varphi(w) \in F &\Leftrightarrow 1 \bullet \varphi(w) = f \in F \\
&\Leftrightarrow [1^H, w, f^H]_{\mathcal{A}} \text{ for some } f \in F \\
&\Leftrightarrow \mathcal{A}(B, \varphi, F) \text{ accepts } w.
\end{aligned}$$

Thus $\varphi^{-1}(F) = L(\mathcal{A}(B, \varphi, F))$. □

It follows from standard arguments that the class Rec of recognizable binoid languages is (effectively) closed under the Boolean operations and inverse homomorphisms, so that if h is a homomorphism $\Sigma^*(\bullet, \circ) \to \Delta^*(\bullet, \circ)$, where Δ is another alphabet, and $L \subseteq \Delta^*(\bullet, \circ)$ is recognizable, then $h^{-1}(L)$ will be recognizable as well.

Corollary 3.39 *The class* Reg *of regular (i.e. recognizable) binoid languages is (effectively) closed under the Boolean operations and inverse homomorphism.*

Remark 3.40 It is clear that $L \subseteq \Sigma^*(\bullet, \circ)$ is recognizable iff there is a finite index congruence θ of $\Sigma^*(\bullet, \circ)$ which *saturates* L, i.e. L is the union of some blocks of the partition induced by θ. Each language $L \subseteq \Sigma^*(\bullet, \circ)$ can be recognized by a smallest binoid called the *syntactic binoid* of L. This binoid B_L, unique up to isomorphisms, corresponds to the *syntactic monoid* [Pin86] of a word language, and to the *syntactic algebra* of a tree language (cf. [Ste98]). For present purposes it is sufficient to define B_L as the quotient of $\Sigma^*(\bullet, \circ)$ with respect to the largest congruence $\sim_L$ which saturates L. We obviously have that L is recognizable iff B_L is finite. The natural homomorphism $\varphi_L : \Sigma^*(\bullet, \circ) \to B_L$ is called the *syntactic homomorphism* of L.

3.10 Rationality

There are several meaningful definitions of rational sets of binoids. Here we will only consider the simplest of them, namely the horizontal rational, vertical rational, birational and generalized birational sets.

3.10.1 Birational and Generalized Birational Languages

Here we will define the classes of birational and generalized birational languages. It will become apparent from the definitions that every birational language is generalized birational, but we will see that the converse inclusion fail.

Recall the closure properties of the recognizable (i.e. regular) class of binoid languages established in Theorem 3.25 and Corollaries 3.26 and 3.39.

Fact 3.41 *The class* $\mathsf{Rec} = \mathsf{Reg}$ *is (effectively) closed under the following operations*

- *Boolean operations (union, intersection, complementation),*
- *horizontal and vertical product,*
- *horizontal and vertical iteration,*
- *homomorphism and inverse homomorphism,*
- ξ*-substitution.*

As usual, a binoid language is called *finite* if it contains a finite number of biwords. Similarly, a binoid language $L \subseteq \Sigma^*(\bullet, \circ)$ is *cofinite* if its complementer with respect to $\Sigma^*(\bullet, \circ)$ is finite. Now we will denote the class of *finite languages* by Fin and the class of *cofinite languages* by CoFin. Next, we will proceed with the definition of birational and generalized birational languages.

Definition 3.42 ([ÉN04]) *The class of* birational languages *is the least class* BRat *of binoid languages containing the finite binoid languages in* $\Sigma^*(\bullet, \circ)$*, for all* Σ*, and closed under union, horizontal and vertical product and horizontal and vertical iteration. The class of* generalized birational languages *is the least class* GRat *of binoid languages which contains the finite languages and is closed under union, horizontal and vertical products, horizontal and vertical iterations and complementation.*

Clearly, $\mathsf{BRat} \subseteq \mathsf{GRat}$. We have already seen in Fact 3.41 that the class Rec of recognizable (i.e. regular) binoid languages is closed under all operations involved in the definition of GRat. Therefore, since every finite language is recognizable, we have that $\mathsf{GRat} \subseteq \mathsf{Rec}$.

In contrast to the case of recognizable tree languages, the recognizable sets of free binoids are not closed under ξ*-iteration* as the following classical example shows:

$$L = \{a \bullet \xi \bullet b\}, \quad L^{*\xi} = \{a^{\bullet n} \bullet \xi \bullet b^{\bullet n} \mid n \geq 0\} \notin \mathsf{Rec}.$$

Here the ξ-iteration of a binoid language is defined as it is for tree languages (cf. [GS84]), i.e. $L^{*\xi} = \cup_{i \geq 0} L_i^{\xi}$, where $L_0^{\xi} = \{\xi\}$, and $L_{i+1}^{\xi} = L_i^{\xi}[L/\xi] \cup L_i^{\xi}$.

Earlier we introduced the concept of bounded depth languages in an informal way. Now we shall state a formal definition for it.

Definition 3.43 ([ÉN04]) *The* alternation depth $\mathrm{ad}(w)$ *of a biword* $w \in \Sigma^*(\bullet, \circ)$ *is defined inductively as follows:*

(i) *If* w *is an empty or singleton biword, then* $\mathrm{ad}(w) = 0$.

(ii) *If* $w = w_1 \bullet \ldots \bullet w_n$, *then* $\mathrm{ad}(w) = \max\{\mathrm{ad}(w_1), \ldots, \mathrm{ad}(w_n)\} + 1$.

(iii) *If* $w = w_1 \circ \ldots \circ w_n$, *then* $\mathrm{ad}(w) = \max\{\mathrm{ad}(w_1), \ldots \mathrm{ad}(w_n)\} + 1$,

In cases (ii) *and* (iii) *the decompositions are maximal and* $n \geq 2$. *The* alternation depth *of a binoid language* L *is defined as the supremum of the alternation depths of its elements:*

$$\mathrm{ad}(L) := \sup\{\, \mathrm{ad}(w) \mid w \in L \,\}.$$

Example 3.44 If $w_1 = a \bullet \langle b \circ c \rangle \bullet d$ and $w_2 = \langle a \circ b \rangle \bullet \langle b \circ \langle c \bullet d \rangle\rangle$, then $\mathrm{ad}(w_1) = 2$, $\mathrm{ad}(w_2) = 3$, $\mathrm{ad}(w_1 \bullet w_2) = 3$ and $\mathrm{ad}(w_1 \circ w_2) = 4$. Note that $\mathrm{ad}(L)$ may be infinite. A recognizable language which has unbounded alternation depth was given in Example 3.19.

We let $\mathsf{BD}^{\leq n}$ stand for the class of binoid languages L with $\mathrm{ad}(L) \leq n$, and let BD stand for the class of *bounded alternation depth* (or just *bounded depth*) *binoid languages*

$$\mathsf{BD} := \bigcup_{n<\infty} \mathsf{BD}^{\leq n}.$$

The next theorem shows that birational languages are just those regular languages that have bounded depth.

Theorem 3.45 ([ÉN04]) $\mathsf{BRat} = \mathsf{Reg} \cap \mathsf{BD}$.

Proof. We have already seen that $\mathsf{BRat} \subseteq \mathsf{GRat}$ and $\mathsf{GRat} \subseteq \mathsf{Reg}$, hence $\mathsf{BRat} \subseteq \mathsf{Reg}$. It is easy to prove that BD is closed under union and the product and iteration operations, hence $\mathsf{BRat} \subseteq \mathsf{BD}$. Thus $\mathsf{BRat} \subseteq \mathsf{Reg} \cap \mathsf{BD}$.

As for the inclusion $\mathsf{Reg} \cap \mathsf{BD} \subseteq \mathsf{BRat}$, we will show by induction on n that for any regular binoid language L we have $L^{[\leq n]} \in \mathsf{BRat}$, for all $n \geq 0$, where $L^{[\leq n]} \subseteq L$ is the set of all biwords in L of alternation depth $\leq n$. Thus if $L \in \mathsf{Reg} \cap \mathsf{BD}$, then there exists an $n \geq 0$ such that $L \in \mathsf{BD}^{\leq n}$, so $L = L^{[\leq n]} \in \mathsf{BRat}$.

Next, assume that $L = L(\mathcal{A})$ for some PA $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$. For any given states $q_1, q_2 \in H$ or $q_1, q_2 \in V$ and $n \geq 0$, let $L^{[\leq n]}_{q_1,q_2}$ denote the language consisting of all biwords of alternation depth at most n on which $\mathcal{A}$ has a run from q_1 to q_2. As

$$L^{[\leq n]} = \bigcup_{i \in I, f \in F} L^{[\leq n]}_{i,f},$$

we conclude our proof by verifying the following lemma.

Lemma 3.46 ([ÉN04]) *For any given states $q_1, q_2 \in H$ or $q_1, q_2 \in V$ and $n \geq 0$, we have $L^{[\leq n]}_{q_1,q_2} \in \mathsf{BRat}$.*

Proof. In the case $n = 0$,

$$L^{[\leq 0]}_{q_1,q_2} = \begin{cases} \{a \mid (q_1, a, q_2) \in \delta\}, & \text{if } q_1 \neq q_2, \\ \{a \mid (q_1, a, q_2) \in \delta\} \cup \{\varepsilon\}, & \text{if } q_1 = q_2. \end{cases}$$

Hence $L^{[\leq 0]}_{q_1,q_2} \in \mathsf{BRat}$.

Note that the case $n = 1$ follows from the classical Kleene theorem for finite words, since only labeling transitions are involved.

Now assume that $L^{[\leq n]}_{s_1,s_2} \in \mathsf{BRat}$, for any s_1 and s_2 horizontal or vertical states of $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$. In order to get a birational expression for $L^{[\leq n+1]}_{q_1,q_2}$ we need to introduce some new notations. We will only consider the case when q_1 and q_2 are horizontal.

By induction, there exists a birational expression $E^{[\leq n]}_{s_1,s_2}$ for each language $L^{[\leq n]}_{s_1,s_2}$. For any two horizontal states s_1, s_2, let HL_{s_1,s_2} denote the ordinary regular language of words accepted by the ordinary automaton with state set H, initial state s_1 and final state s_2, whose set of input letters is the set H^2 with transitions $p_1 \xrightarrow{(p_1,p_2)} p_2$, for all $(p_1, p_2) \in H^2$. Next, let HE_{s_1,s_2} denote an ordinary rational expression for HL_{s_1,s_2}, which exists by the classical Kleene theorem.

When s_1, s_2 are vertical states, define VL_{s_1,s_2} in the same way, and let VE_{s_1,s_2} denote a rational expression for VL_{s_1,s_2}. Let $VE^{++}_{s_1,s_2}$ be rational expressions representing the words of VL_{s_1,s_2} having length at least 2. Now $L^{[\leq n+1]}_{q_1,q_2}$ is given by the expression

$$E^{[\leq n+1]}_{q_1,q_2} = HE_{q_1,q_2}[E^{[\leq n]}_{s_1,s_2}/(s_1, s_2)] \cup \bigcup_{\substack{p_1,p_2 \,:\, \exists \langle\rangle \in \Omega \\ (q_1,\langle,p_1),(p_2,\rangle,q_2) \in \gamma}} VE^{++}_{p_1,p_2}[E^{[\leq n]}_{s_1,s_2}/(s_1, s_2)].$$

Here it is understood that when substituting in HE_{q_1,q_2}, all product operations in HE_{q_1,q_2} are replaced by horizontal product, and all stars by horizontal iteration. Similarly, each product and star in $VE^{++}_{p_1,p_2}$ is replaced by the vertical version of the operation. □

We can even prove that $\mathsf{BRat} \subseteq \mathsf{Reg}_1$. The proof relies on the fact that every birational language has bounded alternation depth. Thus if a PA $\mathcal{A}$ accepts a birational language, then the accepting runs of $\mathcal{A}$ must have a bounded number of opened but not yet closed parentheses at a given moment of the computation process. So we can

construct an equivalent automaton $\mathcal{A}'$ with a single pair of parentheses by storing the information about all such opened parentheses in the (inner) states. This fact together with the previous theorem leads to the following corollary.

Corollary 3.47 ([Ném04]) $\mathsf{BRat} = \mathsf{Reg}_1 \cap \mathsf{BD}$.

This result will be important later in the next section and in Section 3.12.

3.10.2 Horizontal Rational and Vertical Rational Languages

Two subclasses of BRat are also of interest, namely the class of *horizontal rational languages* and the class of *vertical rational languages*. Later we will also define *horizontally bounded* and *vertically bounded* binoid languages and we will study the relationships between the new classes. Horizonal rational and horizontally bounded languages are called *series rational* and *series bounded* in [ÉN02, ÉN04, Ném04]. In much the same way vertical rational and vertically bounded languages are called *parallel rational* and *parallel bounded*.

Definition 3.48 ([ÉN04]) *The class* HRat *of* horizontal rational languages *(or* series rational languages*) is the least class containing the finite languages closed under union, horizontal and vertical products and horizontal iteration.* Vertical rational languages *(or* parallel rational languages*), denoted by* VRat*, are defined symmetrically using vertical iteration instead of horizontal iteration.*

Next, we will define horizontally and vertically bounded languages, but for this we need the notion of the *horizontal and vertical height* of a biword.

Definition 3.49 *The* horizontal height *of a biword w, denoted by* $\mathrm{HHeight}(w)$*, can be computed by induction on the structure of w, in the following way:*

(i) *If $w = \varepsilon$, then* $\mathrm{HHeight}(w) := 0$.

(ii) *If $w = \sigma \in \Sigma$, a singleton biword, then* $\mathrm{HHeight}(w) := 1$.

(iii) *If $w = w_1 \bullet w_2$, a horizontal biword, then*

$$\mathrm{HHeight}(w) := \mathrm{HHeight}(w_1) + \mathrm{HHeight}(w_2).$$

(iv) *If $w = w_1 \circ w_2$, a vertical biword, then*

$$\mathrm{HHeight}(w) := \max\{\, \mathrm{HHeight}(w_1), \mathrm{HHeight}(w_2) \,\}.$$

The vertical height *of w, denoted by* $\mathrm{VHeight}(w)$*, is defined symmetrically.*

Remark 3.50 Note that the notion of horizontal and vertical height is in accordance with the notion of height for posets. Indeed, if the sp-biposet representation of biwords w is $w^{bp} = (P, <_h, <_v, \lambda)$ then $\mathrm{HHeight}(w)$ is the height of the poset $(P, <_h)$, i.e. the length of a maximal $<_h$-chain in $(P, <_h)$. Similarly, $\mathrm{VHeight}(w)$ is the height of the poset $(P, <_v)$.

Definition 3.51 *Call a language $L \subseteq \Sigma^*(\bullet, \circ)$* horizontally bounded *if there is a constant K such that $\mathrm{HHeight}(w) \leq K$ for all $w \in L$, and let HB denote the class of horizontally bounded binoid languages. The class of* vertically bounded *binoid languages, denoted by VB, is defined symmetrically.*

It should be noted here that horizontal rational languages are vertically bounded as VB is closed under the operations used in the definition of HRat. Similarly, vertical rational languages are horizontally bounded. Moreover, if L is birational language which is also vertically bounded, then vertical iteration cannot be used in the construction of L, so L must be horizontal rational. Therefore

$$\mathsf{HRat} = \mathsf{BRat} \cap \mathsf{VB} \text{ and } \mathsf{VRat} = \mathsf{BRat} \cap \mathsf{HB}.$$

Proposition 3.52 ([ÉN04]) $\mathsf{HRat} = \mathsf{Reg} \cap \mathsf{VB}$ *and* $\mathsf{VRat} = \mathsf{Reg} \cap \mathsf{HB}$.

Proof. By Theorem 3.45 $\mathsf{BRat} = \mathsf{Reg} \cap \mathsf{BD}$, and it is not hard to see that $\mathsf{VB} \subseteq \mathsf{BD}$. Indeed, if $\mathrm{VHeight}(w) = K$, then $\mathrm{ad}(w) \leq 2K - 1$. Thus

$$\mathsf{HRat} = \mathsf{BRat} \cap \mathsf{VB} = \mathsf{Reg} \cap \mathsf{BD} \cap \mathsf{VB} = \mathsf{Reg} \cap \mathsf{VB}. \quad \square$$

By Corollary 3.47 we also know that $\mathsf{BRat} = \mathsf{Reg}_1 \cap \mathsf{BD}$, and using this in the proof above we get the following corollary.

Corollary 3.53 $\mathsf{HRat} = \mathsf{Reg}_1 \cap \mathsf{VB}$ *and* $\mathsf{VRat} = \mathsf{Reg}_1 \cap \mathsf{HB}$.

Now we can compare the expressive power of the rational classes introduced in this section.

Theorem 3.54 ([ÉN04])

$$\mathsf{HRat} \cup \mathsf{VRat} \subsetneq \mathsf{BRat} \subsetneq \mathsf{GRat} \subsetneq \mathsf{Reg}.$$

Furthermore, HRat *and* VRat *are incomparable with respect to set inclusion.*

Proof sketch. All the inclusions except the last one are straightforward. Indeed, $\{a\}^{*\bullet} \cup \{a\}^{*\circ} \in \mathsf{BRat} \setminus (\mathsf{HRat} \cup \mathsf{VRat})$, and the language of all biwords, $\Sigma^*(\bullet, \circ)$, is in $\mathsf{GRat} \setminus \mathsf{BRat}$, for any alphabet Σ. On the other hand the proof of $\mathsf{GRat} \subsetneq \mathsf{Reg}$ is rather complicated. The proof rely on a generalization of Schützemberger's theorem, that states that if a word language is star-free then its syntactic monoid if aperiodic [Sch65]. See [ÉN04] for the details about this.

3.10.3 Some Decidability Results

Our next aim is to show that it can be decided whether a regular binoid language is finite, cofinite, birational, horizontal rational or vertical rational. At present we do not know the answer for generalized birational languages, however. In this section we will follow [ÉN04] and work with binoid languages that do not contain the empty biword ε. Then we can use syntactic bisemigroups instead of syntactic binoids. This is not a real restriction, as L is birational, horizontal or vertical rational if and only if $L \setminus \{\varepsilon\}$ has this property. But excluding the empty biword guarantees, for example, that $\mathrm{HHeight}(w_1 \bullet w_2) > \mathrm{HHeight}(w_1)$ for all $w_1, w_2 \in \Sigma^+(\bullet, \circ)$. Again we need to do some preparatory work before we can prove the above results.

Definition 3.55 ([ÉN04]) *Let $(B, \bullet, \circ)$ be an arbitrary bisemigroup and suppose that $p, q \in B$. We will denote the fact that q is a* horizontal (vertical) factor *of p by*

$$p \succ_h q \quad \Leftrightarrow \quad \exists\, r, s \in B \ : \ p = r \bullet q \bullet s \text{ or } p = r \bullet q \text{ or } p = q \bullet s;$$
$$p \succ_v q \quad \Leftrightarrow \quad \exists\, r, s \in B \ : \ p = r \circ q \circ s \text{ or } p = r \circ q \text{ or } p = q \circ s;$$

Since the operations $\bullet$ and $\circ$ are associative, the relations $\succ_h$ and $\succ_v$ are transitive.

Next, the following facts are clear:

Lemma 3.56 ([ÉN04])

(i) If $\varphi : B_1 \to B_2$ is a homomorphism of bisemigroups and $p, q \in B_1$, then $p \succ_h q \Rightarrow \varphi(p) \succ_h \varphi(q)$;

(ii) If $\varphi : B_1 \to B_2$ is a surjective homomorphism of bisemigroups and $p', q' \in B_2$, then $p' \succ_h q' \Rightarrow \forall q \in \varphi^{-1}(q')\ \exists p \in \varphi^{-1}(p') \ : \ p \succ_h q$.

Similar statements hold for $\succ_v$.

Definition 3.57 ([ÉN04]) *Suppose that $x_1, x_2, \ldots$ is a finite or infinite sequence of elements of a bisemigroup. For all $i \geq 1$, let $\succ_i$ be the relation $\succ_h$ or the relation $\succ_v$. When $x_i \succ_i x_{i+1}$ holds for each $i \geq 1$, we call the sequence $x_1 \succ_1 x_2 \succ_2 x_3 \succ_3 \ldots$ a* composition chain. *Moreover, if $\succ_i = \succ_h$ for an infinite number of indices i, then we call the chain* horizontally infinite. Vertically infinite *composition chains are defined symmetrically. Finally, a chain is an* alternating composition chain *if for all n, $\succ_n = \succ_h$ and $\succ_{n+1} = \succ_v$, or $\succ_n = \succ_v$ and $\succ_{n+1} = \succ_h$. (E.g. $x_1 \succ_h x_2 \succ_v x_3 \succ_h x_4 \succ_v x_5$ is an alternating composition chain.)*

Proposition 3.58 ([ÉN04]) *For any $L \in \Sigma^+(\bullet, \circ)$, let B_L denote the syntactic bisemigroup of L and let $\varphi_L : \Sigma^+(\bullet, \circ) \to B_L$ denote the corresponding syntactic homomorphism. (See Remark 3.40.) Then,*

$$\begin{aligned}
L \in \mathsf{Fin} &\Leftrightarrow B_L \text{ is finite and there is no infinite composition chain} \\
&\quad \text{in } B_L \text{ starting from some element of } \varphi_L(L); \\
L \in \mathsf{CoFin} &\Leftrightarrow B_L \text{ is finite and there is no infinite composition chain} \\
&\quad \text{in } B_L \text{ starting from some element of } B_L \setminus \varphi_L(L); \\
L \in \mathsf{HRat} &\Leftrightarrow B_L \text{ is finite and there is no vertically infinite composition chain} \\
&\quad \text{in } B_L \text{ starting from some element of } \varphi_L(L); \\
L \in \mathsf{VRat} &\Leftrightarrow B_L \text{ is finite and there is no horizontally infinite composition chain} \\
&\quad \text{in } B_L \text{ starting from some element of } \varphi_L(L); \\
L \in \mathsf{BRat} &\Leftrightarrow B_L \text{ is finite and there is no infinite alternating composition chain} \\
&\quad \text{in } B_L \text{ starting from some element of } \varphi_L(L).
\end{aligned}$$

Proof. All statements are quite self-evident. For example, for the first two claims, one observes that, for every finite bisemigroup B and $b \in B$, there is no infinite composition chain starting from b if and only if there is a constant K such that all composition chains are of length at most K. For the last three claims one also uses Theorem 3.45 and Proposition 3.52. □

Theorem 3.59 ([ÉN04]) *It is decidable whether a regular binoid language is finite, cofinite, birational, horizontal rational or vertical rational.*

Proof. For any regular (hence recognizable) binoid language L one can compute the syntactic bisemigroup of $L \setminus \{\varepsilon\}$. Since the syntactic bisemigroup is finite it is decidable whether $p \succ_h q$ or $p \succ_v q$ holds for any two elements of it. Hence the existence of the infinite composition chains in Proposition 3.58 is also decidable. □

3.11 Logical Definability

In this section we will relate monadic second-order definable (MSO-definable) binoid languages to recognizable languages. It is an important and actively investigated general question of theoretical computer science whether, for certain classes of structures, MSO-definability and recognizability are equivalent. Known equivalences includes languages of finite and infinite words [Büc60, Elg61], finite trees [Don70, TW68], traces [Tho90],

sp-posets [Kus03a], texts of bounded primitivity [HtP97] and graphs under various finiteness conditions [Lap98, Cou03, Weil04b], see also [Wei04a].

In this section, by relying on the main result of Hoogeboom and ten Pas [HtP97] on text languages, we will prove that the equivalence of recognizability and MSO-definability holds for binoid languages as well.

3.11.1 MSO-definable Binoid Languages

We will start with some basic definitions of logical definability. For extending logical definability to binoid languages the easiest way is to regard biwords as relational structures where the horizontal and vertical order relations are explicitly present. So here we will consider any biword $w \in \Sigma^*(\bullet, \circ)$ in its sp-biposet representation $w^{bp} = (P, <_h, <_v, \lambda)$ (cf. Section 2.6.2).

Now suppose that Σ is an alphabet. An *atomic formula* is of the form $Q_a(x)$, $X(x)$, $x <_h y$ or $x <_v y$, where a is any letter in Σ, x, y are *first-order variables* ranging over vertices in an sp-biposet, while X is a *monadic second-order variable* ranging over subsets of the vertex set of an sp-biposet. Here $Q_a(x)$ means that vertex x is labeled by a and $X(x)$ means that x belongs to X. The atomic formulas $x <_h y$ and $x <_v y$ have their expected meanings. (We assume a fixed countable set of first-order, and a fixed countable set of second-order variables.) *Formulas* are composed from atomic formulas by the Boolean connectives $\vee$ and $\neg$ and first- and second-order existential quantifiers $\exists x$ and $\exists X$. We define in the usual way when a *closed formula (sentence) φ holds in, or is satisfied by an sp-biposet w*, denoted $w \models \varphi$. The *language L_φ defined by φ* is

$$L_\varphi := \{\, w \in \Sigma^*(\bullet, \circ) \mid w \models \varphi \,\}.$$

Definition 3.60 ([ÉN04]) *We say that a language $L \subseteq \Sigma^*(\bullet, \circ)$ is* MSO-definable *if there is a sentence φ with $L = L_\varphi$. We let* MSO *denote the class of* MSO*-definable languages in $\Sigma^*(\bullet, \circ)$, for all alphabets Σ.*

Remark 3.61 For relational structures in general, besides the MSO logic it is usual to consider the so-called *counting monadic second-order logic*. This means that we enrich the MSO logic with first order *modulo quantifiers*, which are of the form $\exists x^{\bmod q}$, where $q > 1$ is an integer. The interpretation of such a quantifier is that the number of positions x that satisfy the subformula that has been quantified, is divisible by q. For graphs and graph-like structures the counting second order monadic definable class of structures (denoted by CMSO) can be larger than MSO, and it seems that CMSO is a more natural counterpart of the concept of algebraic recognizability [Cou91, Cou03, CW05, Weil04b]. On the other hand if one can define a linear order on the domains of the structure by an MSO-formula – and obviously this can be done in the case of biwords –, then CMSO is equal to MSO.

It is not hard to show that $\mathsf{MSO} \subseteq \mathsf{Rec}$. We can argue by formula induction. In order to do this, we first associate a language $L_\varphi \subseteq \mathrm{SPB}(\Sigma \times \mathcal{V} \times \mathcal{P}(\mathcal{W}))$ with any formula φ whose free variables are contained in the finite sets $\mathcal{V}$ of first-order and $\mathcal{W}$ of second-order variables, where $\mathcal{P}(\mathcal{W})$ denotes the powerset of $\mathcal{W}$. Our definition parallels that given in [Str94], and makes use of the closure properties of recognizable languages given in Fact 3.41. (An alternative way of proving $\mathsf{MSO} \subseteq \mathsf{Rec}$ would be via a compositionality property of the monadic theories of sp-biposets. See Kuske [Kus03a] for a general outline of this method.)

3.11.2 Texts and Text Languages

In order to prove that even $\mathsf{MSO} = \mathsf{Rec}$ holds for binoid languages, we should realize that binoids are special cases of texts introduced by Ehrenfeucht and Rozenberg [ER93]. Texts themselves are special cases of 2-structures [ER90, ER92, EHPR96]. Recognizable and MSO-definable text languages were first studied by Hoogeboom and ten Pas in [HtP96, HtP97]. Here we will just present those concepts about texts which are needed to relate them with sp-biposets. For more details the reader should see [ER93, EPR93, HtP96, HtP97]. In addition, here we will use sightly different notations from those used in these articles to make them similar to our notations for biposets.

Definition 3.62 *A* text *(over the alphabet Σ) is a 4-tuple $(P, \rho_1, \rho_2, \lambda)$, where P is a set, ρ_1 and ρ_2 are two strict linear order relations on P, and $\lambda : P \to \Sigma$ is a labeling function.*

As in the case of biposets, we will not distinguish between isomorphic texts, hence we may assume that $P = \{1, 2, \ldots, n\}$, for some $n \geq 0$, and ρ_1 is the standard order $1 < 2 < \ldots < n$. This form will be called the *standard form* of a text, as in [ER93, HtP96, HtP97]. If a text $\tau = (P, \rho_1, \rho_2, \lambda)$ is in standard form and $P = [n]$, then τ can also be written as $\tau = (\lambda(1)\lambda(2)\ldots\lambda(n), \rho_2)$.

Example 3.63 Assume that $\Sigma = \{\, a, b \,\}$, $P := [6]$, $\rho_1 = (1, 2, 3, 4, 5, 6)$, $\rho_2 = (3, 1, 4, 2, 6, 5)$ and $\lambda : P \to \Sigma$ with $\lambda(1) = \lambda(3) = \lambda(5) = a$ and $\lambda(2) = \lambda(4) = \lambda(6) = b$. Here linear orders are represented as usual, so $(i_1, i_2, \ldots, i_n)$ means the linear order $i_1 < i_2 < \ldots < i_n$. Now $\tau = (P, \rho_1, \rho_2, \lambda)$ is a text. As τ is in normal form it can also be written as $\tau = (ababab, (3, 1, 4, 2, 6, 5))$.

If we forget about the labeling of a text we get the notion of a *bi-order*. Thus a bi-order is a pair of two strict linear orders over a common domain. It will be denoted

by (P, ρ_1, ρ_2), or just by ρ_2, if it is in standard form i.e. when $P = [n]$, and $\rho_1 = (1, 2, \ldots, n)$.

Suppose that ρ is a bi-order in standard form on an n element set, and $\tau_1 \ldots \tau_n$ are texts over a common alphabet Σ. Then it is natural to define the *substitution* of τ_1,τ_2, $\ldots$, τ_n into ρ. The result of the substitution is a text $[\rho \leftarrow (\tau_1, \ldots, \tau_n)]$ which is a disjoint union of the texts τ_i-s that also inherits the orders determined by ρ. See [HtP97] for more details and examples. The substitution of bi-orders into a bi-orders can be defined in the same way.

We say that a text (resp. bi-order) is *decomposable* if it can be obtained as a substitution of some proper subtexts (resp. bi-orders) into a bi-order. Texts and bi-orders which are not decomposable are called *primitive*. Obviously, on the two-element set $\{1, 2\}$ both bi-orders are primitive. They are denoted (in standard form) by $\sigma_f = (1, 2)$ and $\sigma_b = (2, 1)$, and called *forward sequential* and *backward sequential* bi-orders, respectively. But it turned out that there exist arbitrary large primitive bi-orders and texts [ER90, ER93].

Example 3.64 In our example above $\tau = [\sigma_f \leftarrow (\tau_1, \tau_2)]$, where $\tau_1 = (abab, (3, 1, 4, 2))$ is a primitive text, but τ_2 can be further decomposed into $\tau_2 = [\sigma_b \leftarrow (\mathbf{a}, \mathbf{b})]$. Here, of course, $\mathbf{a}$ and $\mathbf{b}$ stand for the singleton texts $\mathbf{a} = (a, (1))$ and $\mathbf{b} = (b, (1))$.

The class of MSO-definable text languages is defined in just the same way as the MSO-definable binoid languages but we have to use the atomic formulas $x \sqsubset_1 y$ and $x \sqsubset_2 y$ instead of $x <_h y$ and $x <_v y$. Needless to say, they will be interpreted as the two linear order relations of a text.

On the other hand defining a suitable algebraic framework for texts in general is not so obvious (see [HtP96]). If ρ is a primitive bi-order on n elements, then we can naturally associate with it an n-ary operation π_ρ on texts like so:

$$\pi_\rho(\tau_1, \ldots, \tau_n) := [\rho \leftarrow (\tau_1, \ldots, \tau_n)].$$

A problem arises from the fact that there are infinitely many primitive bi-orders. Hence, unless we wish to work with algebras of infinitely many operations, we need to restrict the bi-orders that can be used to construct texts from the singleton texts. Thus we arrive at the notion of *bounded primitivity*. Now suppose that Δ is an alphabet and Π is a finite set of primitive bi-orders. Let $\mathrm{TXT}_\Pi(\Delta)$ denote the set of those texts over Δ which can be constructed from the singleton text (corresponding to the letters of Δ) by the π_ρ operations with $\rho \in \Pi$. We say that a text language L is of bounded primitivity if it is a subset of $\mathrm{TXT}_\Pi(\Delta)$ for some finite Π and Δ. Now $\mathrm{TXT}_\Pi(\Delta)$ is an algebra

under the operation of π_ρ-s ($\rho \in \Pi$), and recognizability can be defined in the usual way. One of the main results of [HtP97] is that for text languages of bounded primitivity recognizability is equivalent to MSO-decidability.

But we only need this result in a special case of *uniformly nonprimitive* (also called *alternating*) *texts* [ER93]. The set of alternating texts is defined as $\mathrm{ATXT}(\Delta) := \mathrm{TXT}_{\{\sigma_\mathrm{f},\sigma_\mathrm{b}\}}(\Delta)$. For simplicity let denote the two operations of alternating texts by

$$\tau_1 \oplus \tau_2 := [\sigma_f \leftarrow (\tau_1, \tau_2)] \text{ and}$$
$$\tau_1 \otimes \tau_2 := [\sigma_b \leftarrow (\tau_1, \tau_2)].$$

Of course we can extend these operations to text languages in a natural way. Note that both $\oplus$ and $\otimes$ are associative, hence $(\mathrm{ATXT}(\Delta), \oplus, \otimes)$ is a bisemigroup, or a binoid, if we also include the empty text. As was mentioned previously, we need the following result from [HtP97], namely:

Theorem 3.65 (Hoogeboom and ten Pas) *For all $L \subseteq \mathrm{ATXT}(\Delta)$, L is recognizable if and only if L is* MSO*-definable.*

3.11.3 Sp-biposets and Alternating Texts

We now turn to establish a connection between biposets and texts. First recall that $P = (P, <_h, <_v, \lambda_P)$ is a biposet, if $<_h$ and $<_v$ are partial order relations on a set P, and λ is a labeling function $\lambda : P \to \Sigma$. Recall that a biposet P is total if any two elements of P are related by exactly one of the two partial orders. We will show that there is a bijective correspondence between labeled total biposets and texts, and also that this bijection respects the operations, hence it defines an isomorphism.

Definition 3.66 *Let $P = (P, <_h, <_v, \lambda_P)$ be a total biposet, and let its* text representation *P^{txt}, be defined by $P^{txt} := (P, \sqsubset_1, \sqsubset_2, \lambda_P)$, where*

$$\sqsubset_1 := <_h \cup <_v, \qquad \sqsubset_2 := <_h \cup <_v^{-1} \tag{3.3}$$

and $<_v^{-1}$ is the reverse of the relation $<_v$. Let txt *denote the function* $\mathrm{txt} : P \mapsto P^{txt}$.

Lemma 3.67 *P^{txt} is a text for each total biposet P. Moreover,* txt *is a bijective function between total biposets and texts.*

Proof. To see that P^{txt} is a text, we need to show that both $\sqsubset_1$ and $\sqsubset_2$ are linear orders on P.

The irreflexivity of $\sqsubset_1$ and $\sqsubset_1$ is naturally follows from the irreflexivity of $<_h$ and $<_v$. As for the transitivity, suppose that $x \sqsubset_1 y$ and $y \sqsubset_1 z$ for some x, y and z in P. From the totality of P it follows that in the biposet x and z are related by one of the relations $<_h$, $<_v$, $<_h^{-1}$ or $<_v^{-1}$. But it can be readily verified that $x <_h^{-1} z$ or $x <_v^{-1} z$ would contradict the transitivity of $<_h$ and $<_v$. Hence $x <_h z$ or $x <_v z$, so $x \sqsubset_1 z$. The transitivity of $\sqsubset_2$ can be obtained in a similar way, replacing $<_h$ by $<_h^{-1}$, which is also transitive.

Furthermore, the relations $<_h$ and $<_v$ can be recovered from $\sqsubset_1$ and $\sqsubset_2$ in the following way:

$$<_h = \sqsubset_1 \cap \sqsubset_2, \qquad <_v = \sqsubset_1 \cap \sqsubset_2^{-1}. \tag{3.4}$$

Hence the correspondence $P \mapsto P^{txt}$ is injective.

Next, it is straightforward to see that for any two linear orders $\sqsubset_1$ and $\sqsubset_2$, equations (3.4) define two partial orders on P, such that any two elements of P are related by exactly one of them. Therefore the correspondence $P \mapsto P^{txt}$ is also surjective, hence it is a bijection between total biposets and texts. □

Remark 3.68 Besides totality sp-biposets must also satisfy the N-free condition of Corollary 2.27 for both partial order relations. This translates to the "primitive quartet-freeness" property of texts [ER92, EPR93].

As a direct consequence of the definitions of $\bullet$, $\circ$, σ_f, σ_b and txt, we get

$$(P_1 \bullet P_2)^{txt} = P_1^{txt} \oplus P_2^{txt} \text{ and } (P_1 \circ P_2)^{txt} = P_1^{txt} \otimes P_2^{txt}.$$

Since sp-biposets are defined as those biposets that can be built from the singleton biposets using the horizontal and vertical product operations, they correspond to the alternating texts, hence we have:

Lemma 3.69 *For any alphabet Σ the mapping* txt *is an isomorphism between the free binoid $\Sigma^*(\bullet, \circ)$ and alternating texts $(\mathrm{ATXT}(\Sigma), \otimes, \oplus)$. Moreover, the extension of* txt *to languages is an isomorphism between binoid languages and alternating text languages.*

Since equations (3.3) and (3.4) can be expressed by first-order formulas, a binoid language L is second-order definable iff the corresponding alternating text language $\mathrm{txt}(L)$ is second-order definable. Also, from Lemma 3.69 the notion of recognizability is the same for binoid and alternating text languages. Thus from Theorem 3.65 we immediately have:

Theorem 3.70 ([ÉN04]) *Any binoid language is recognizable, if and only if it is* MSO*-definable, i.e.* Rec = MSO.

In Chapter 4 we shall supply another proof of Theorem 3.70, which will form the basis of the generalization to the infinite case.

From Theorem 3.35, Theorem 3.45 and Theorem 3.70 we can derive the following corollary.

Corollary 3.71 ([ÉN04]) *The following conditions are equivalent for a language $L \subseteq \Sigma^*(\bullet, \circ)$ of bounded depth:*

1. *L is recognizable.*
2. *L is regular.*
3. *L is birational.*
4. *L is generalized birational.*
5. *L is MSO-definable.*

When L is vertically bounded, the above conditions are further equivalent to the condition that L is horizontal rational.

3.12 Comparison with the Monoid Approach of Hashiguchi et al.

An independent study on automata and regularity of binoid languages was carried out by Hashiguchi et al. in a series of papers [HIJ00, HWJ03, HSJ04]. They used ordinary finite automata (which we shall call *monoid automata* from now on) operating on the term representations of biwords. Actually they defined two classes of regular binoid languages. The aim of this section is to relate this approach to our PA and the classes Reg_i. We will find that their notion of regularity is less general than ours. Moreover, we will show how their monoid approach can be extended to our broader class of regular binoid languages (Reg).

3.12.1 The Monoid Approach of Hashiguchi et al.

In [HIJ00] Hashiguchi et al. introduced two modes of operations of monoid automata for defining regular binoid languages. Recall that $E(\Sigma) := \Sigma \cup \{ \bullet, \circ, \langle, \rangle \}$.

Definition 3.72 ([HIJ00]) *Given a monoid automaton $\mathcal{A}$ over the alphabet $E(\Sigma)$ and a binoid language $L \subseteq \Sigma^*(\bullet, \circ)$, we say that*
i) $\mathcal{A}$ accepts *L* in the free monoid mode *if, for any word $x \in E(\Sigma)^*$, $\mathcal{A}$ accepts x iff x is the term representation of a biword in L;*

ii) $\mathcal{A}$ accepts L in the free binoid mode *if, for any term representation* $w \in \mathrm{TM}(\Sigma)$, $\mathcal{A}$ *accepts* w *iff* w *is the term representation of a biword in* L.

Let $\mathsf{Reg}^{\mathsf{FM}}$ and $\mathsf{Reg}^{\mathsf{FB}}$ denote the classes of binoid *languages that can be accepted in the free monoid mode* and *in the free binoid mode*, respectively.

The concept of relativized regularity below will be useful for giving brief formulations of various acceptance modes.

Definition 3.73 ([Ném07]) *Let* Σ *be an alphabet and* $U \subseteq \Sigma^*$ *be an arbitrary language. Now consider a language* $L \subseteq U$. *We say that* L *is* regular relative to U *(or* L *is* U-regular *for short), if there exists a regular language* $\hat{L} \subseteq \Sigma^*$ *such that* $L = \hat{L} \cap U$.

For example $L = \{\langle^n\rangle^n \mid n \geq 0\}$ is a Dyck-regular language, since $L = \hat{L} \cap D_1$ with a regular language $\hat{L} = \langle^*\rangle^*$, where D_1 denotes the Dyck language over a single pair of parentheses (cf. [Har78]). We can now reexpress Definition 3.72 in the following way.

Fact 3.74 (i) $L \in \mathsf{Reg}^{\mathsf{FM}} \Leftrightarrow L^{tm}$ *is a regular word language.*

(ii) $L \in \mathsf{Reg}^{\mathsf{FB}} \Leftrightarrow L^{tm}$ *is* $\mathrm{TM}(\Sigma)$*-regular word language.*

It is not hard to see that $\mathsf{Reg}^{\mathsf{FM}} \subseteq \mathsf{BD}$ and $\Sigma^*(\bullet, \circ) \in \mathsf{Reg}^{\mathsf{FB}} \setminus \mathsf{BD}$. Hence we have $\mathsf{Reg}^{\mathsf{FM}} \subsetneq \mathsf{Reg}^{\mathsf{FB}}$ (cf. [HIJ00]). The main result of [HWJ03, HSJ04] can be summarized as follows.

Theorem 3.75 (Hashiguchi et al. [HWJ03, HSJ04]) $\mathsf{Reg}^{\mathsf{FM}} = \mathsf{BRat}$.

The result above gives a nice operational characterization of $\mathsf{Reg}^{\mathsf{FM}}$, but this class is not closed under complementation, because $\mathsf{BRat} \neq \mathsf{GRat}$ by Theorem 3.54.

We can build another acceptance mode of monoid automata by using the cterm representation instead of terms. We say that a monoid automaton $\mathcal{A}$ *accepts a binoid language* L *in the* C_1*-mode* if, for any cterm representation $w \in \mathrm{CTM}(\Sigma)$, $\mathcal{A}$ accepts w iff w is the cterm representation of a biword in L. From now on the free binoid mode will also be called the T_1*-mode*. Moreover, we will extend the T_1 and C_1 modes using i-terms and i-cterms.

Definition 3.76 ([Ném07]) *Given an integer* $i \geq 0$, *a monoid automaton* $\mathcal{A}$ *over the alphabet* $E_i(\Sigma)$ *and a binoid language* $L \subseteq \Sigma^*(\bullet, \circ)$, *we say that* $\mathcal{A}$ accepts L in the T_i-mode *(resp.* in the C_i-mode*) if, for any biword* $w \in \Sigma^*(\bullet, \circ)$, $w \in L$ *iff* $\mathcal{A}$ *accepts at least one* i*-term (resp.* i*-cterm) representation of* w.

Let $\mathsf{Reg}_i^{\mathsf{T}}$ and $\mathsf{Reg}_i^{\mathsf{C}}$ denote the classes of languages that can be accepted by a monoid automaton over $E_i(\Sigma)$ in the T_i-mode and the C_i-mode, respectively. Furthermore, let $\mathsf{Reg}_\infty^{\mathsf{T}} := \cup_{i=0}^{\infty}\mathsf{Reg}_i^{\mathsf{T}}$ and $\mathsf{Reg}_\infty^{\mathsf{C}} := \cup_{i=0}^{\infty}\mathsf{Reg}_i^{\mathsf{C}}$.

Fact 3.77

(i) $L \in \mathsf{Reg}_i^{\mathsf{T}} \Leftrightarrow$ *there exists a* $\mathrm{TM}_i(\Sigma)$*-regular* i*-term representation of* L.

(ii) $L \in \mathsf{Reg}_i^{\mathsf{C}} \Leftrightarrow$ *there exists a* $\mathrm{CTM}_i(\Sigma)$*-regular* i*-cterm representation of* L.

3.12.2 A Comparison

In this section we present the main result of the section, namely the equivalence of PA that have i pairs of parentheses with monoid automata in both the T_i-mode and C_i-mode. For this we need to slightly modify the acceptance conditions of PA. Namely, we will not allow indirect runs as accepting runs. If $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ is a PA, let $L'(\mathcal{A}) := \{\,\mathrm{Label}(\mathbf{r}) \mid \mathbf{r} \in \mathrm{Runs}(\mathcal{A}), \mathrm{start}(\mathbf{r}) \in I, \mathrm{end}(\mathbf{r}) \in F$ and $\mathbf{r}$ *is not an indirect run*$\}$ and additionally, if $I \cap F \neq \emptyset$ then $L'(\mathcal{A})$ also contains ε, the empty biword. Moreover, let Reg_i' denote the class of those binoid languages that can be written as $L'(\mathcal{A})$ with a PA that has at most i pairs of parentheses.

At first sight it seems that the classes Reg_i' are smaller than the original classes Reg_i. But this is not so; on the contrary, while indirect acceptance can be simulated in the new "no indirect acceptance" mode, the converse simulation is not possible. If we have a PA $\mathcal{A}$ whose initial and final states are all horizontal, then we are certain that $L'(\mathcal{A})$ just contains horizontal biwords. On the other hand, this property cannot be guaranteed in the old acceptance mode. Thus one can verify (with considerable effort) the following correspondence between the new and the old acceptance modes of PA.

Theorem 3.78 ([Ném07]) *We have* $\mathsf{Reg}_0 = \mathsf{Reg}_0'$*, and* $\mathsf{Reg}_i \subsetneq \mathsf{Reg}_i' \subsetneq \mathsf{Reg}_{i+1}$*, for all* $i \geq 1$.

Proof. The equality $\mathsf{Reg}_0 = \mathsf{Reg}_0'$ is trivial. We will show in turn that $\mathsf{Reg}_i \subseteq \mathsf{Reg}_i'$, $\mathsf{Reg}_i' \subseteq \mathsf{Reg}_{i+1}$, $\mathsf{Reg}_i \neq \mathsf{Reg}_i'$ and $\mathsf{Reg}_i' \neq \mathsf{Reg}_{i+1}$.

The first two inclusions can be seen by direct modifications of PA. For $\mathsf{Reg}_i \subseteq \mathsf{Reg}_i'$ the indirect accepting runs of an automaton $\mathcal{A}$ can be simulated in the new mode by adding disjoint copies of $\mathcal{A}$ to itself for each pseudo initial-final pair of states (cf. the proof of Lemma 3.23), and marking these states as initial and final states.

To verify that $\mathsf{Reg}_i' \subseteq \mathsf{Reg}_{i+1}$, consider a PA $\mathcal{A}$ that has i pairs of parentheses and works in the new "no indirect acceptance" mode. One can define an equivalent

automaton to $\mathcal{A}$ with an additional $i+1$th pair of parentheses working in the original mode. For this add new states i_*^h, f_*^h, i_*^v and f_*^v to $\mathcal{A}$, and apply the new $i+1$th pair of parentheses to connect them to all the original initial and final states of $\mathcal{A}$. Of course, let $\{i_*^h, i_*^v\}$, and $\{f_*^h, f_*^v\}$ to be the sets of initial and final states, resp. This will ensure that the modified automaton will have indirect accepting runs only, but the "inner parts" of these runs are simply the direct accepting runs of $\mathcal{A}$. On the other hand, the labels of the indirect (hence nonaccepting) runs of $\mathcal{A}$ from an initial to a final state will not be in the language of $\mathcal{A}'$ as double-parenthesization is forbidden.

Next, we will show that $\mathsf{Reg}_1 \neq \mathsf{Reg}_1'$. A language which separates these two classes is, for instance, $\mathrm{Hor}(\Sigma)$ – the set of all horizontal biwords over Σ. Indeed, $\mathrm{Hor}(\Sigma)$ is in Reg_1', since the automaton depicted in Figure 3.2 in "no indirect acceptance" mode accepts precisely $\mathrm{Hor}(\Sigma)$. On the other hand, we will prove that $\mathrm{Hor}(\Sigma) \notin \mathsf{Reg}_1$. Suppose that, on the contrary, there is a PA $\mathcal{A}$ with a single pair of parentheses that accepts $\mathrm{Hor}(\Sigma)$ in the original mode.

Now assume that a is in Σ and consider biwords whose cterm representation is of the form

$$\bullet\, [\langle a]^k [a\rangle]^k [\langle a]^k [a\rangle]^k, \qquad \text{where } k \geq 1.$$

Since $\mathcal{A}$ has a finite number of states it is possible to find integers $k_1 > k_2$ such that $\mathcal{A}$ accepts both $\bullet\, [\langle a]^{k_1} [a\rangle]^{k_1} [\langle a]^{k_1} [a\rangle]^{k_1}$ and $\bullet\, [\langle a]^{k_2} [a\rangle]^{k_2} [\langle a]^{k_2} [a\rangle]^{k_2}$; moreover during the processing of these two biwords $\mathcal{A}$ is in the same states at the beginning, at the end, and also among the four basic blocks : $[\langle a]^i$, $[a\rangle]^i$, $[\langle a]^i$ and $[a\rangle]^i$, for $i = k_1, k_2$. Let $q_1, q_2, \ldots, q_5$ denote these states in turn. Now as $\mathcal{A}$ has a single pair of parentheses it also has a run from q_1 to q_5 whose label is $[\langle a]^{k_1} [a\rangle]^{k_2} [\langle a]^{k_2} [a\rangle]^{k_1}$. But now $k_1 > k_2$ implies that this is an indirect run of $\mathcal{A}$ (starting and ending with a matching parenthesizing transition pair), hence at least one vertical biword is also accepted by $\mathcal{A}$, contradicting our assumption which states that $L(\mathcal{A}) = \mathrm{Hor}(\Sigma)$. In much the same way one can show that $\mathsf{Reg}_i \neq \mathsf{Reg}_i'$.

Finally $\mathsf{Reg}_i' \neq \mathsf{Reg}_{i+1}$ comes from the observation that the language $\tilde{L}(\Sigma_{i+1})$ – given in the proof of Theorem 3.32 and separating Reg_i and Reg_{i+1} – also belongs to Reg_i'. □

Earlier results, namely Theorem 3.45 and Theorem 3.75, lead to the following characterization of the free monoid mode in terms of PA.

Theorem 3.79 ([Ném07]) $\mathsf{Reg}^{\mathsf{FM}} = \mathsf{Reg}_1 \cap \mathsf{BD} = \mathsf{Reg}_1' \cap \mathsf{BD}$.

Now we will establish a connection between the free binoid mode and PA.

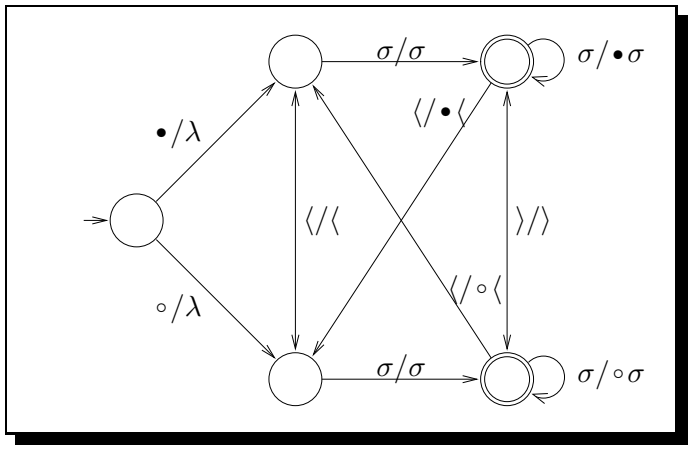

Figure 3.9: *A finite transducer which transforms* 1*-cterms into* 1*-terms over a one-letter alphabet* $\Sigma = \{\sigma\}$.

Lemma 3.80 ([Ném07]) *For any* $i \geq 0$*, there exists a finite transduction* $\tau_i : E_i(\Sigma)^* \to E_i(\Sigma)^*$ *which transforms every* i*-cterm to the equivalent* i*-term.*

Proof sketch. Observe that τ_i can be induced by a finite transducer like the one depicted in Figure 3.9.

Corollary 3.81 ([Ném07]) *Suppose that* $L \subseteq \mathrm{CTM}_i(\Sigma)$ *is a word language for some* $i \geq 0$*. Then* L *is* $\mathrm{CTM}_i(\Sigma)$*-regular iff* $\tau_i(L)$ *is* $\mathrm{TM}_i(\Sigma)$*-regular.*

This result can be interpreted as follows. In the description of a binoid language by words we can use the cterm (resp. i-cterm) representation instead of the term (resp. i-term) representation, i.e. we can neglect the operation symbols without affecting the regularity of the language. This simplification may be useful in syntactic proofs using representations of biwords via words as in [HWJ03] and [HSJ04], and it is also crucial in the proof of our main theorem presented below.

Theorem 3.82 ([Ném07]) *For any integer* $i \geq 0$*, we have* $\mathsf{Reg}'_i = \mathsf{Reg}_i^{\mathsf{C}} = \mathsf{Reg}_i^{\mathsf{T}}$.

Proof sketch. The second equality can easily be derived from Corollary 3.81. On the other hand, the proof of $\mathsf{Reg}'_i = \mathsf{Reg}_i^{\mathsf{C}}$ is rather technical. We need to transform a PA into an equivalent monoid automaton over $E_i(\Sigma)$ and vice versa. We will use the following notation for a (nondeterministic) monoid automaton: $\mathcal{A} = (S, \Sigma, \delta, I, F)$, where S is the set of states, Σ is the input alphabet, $\delta : S \times \Sigma \to 2^S$ is the transition function, and I and F are the sets of initial and final states, respectively.

Let $\mathcal{A} = (S, H, V, \Sigma, \Omega_i, \delta, \gamma, I, F)$ be a PA. If we do not distinguish between the horizontal and vertical states, and if we do not distinguish between labeling and parenthesizing transitions, then we obtain a monoid automaton $(S, \Sigma \cup \Omega_i, \delta \cup \gamma, I, F)$. Let us take two new states $i_*, f_* \notin S$. Now replace I with $\{i_*\}$, and add the following transitions to δ

$$\delta' = \{(i_*, \bullet, i) \mid i \in I \cap H\} \cup \{(i_*, \circ, i) \mid i \in I \cap V\} \cup \{(i_*, \sigma, f_*) \mid \sigma \in L'(\mathcal{A})\}.$$

We will regard i_* as a final state, iff $\varepsilon \in L'(\mathcal{A})$. Thus we get a monoid automaton $\mathcal{A}^M := (S, E_i(\Sigma), \delta \cup \gamma \cup \delta', \{i_*\}, F')$, where $F' = F \cup \{i_*, f_*\}$, if $\varepsilon \in L'(\mathcal{A})$, and $F' = F \cup \{f_*\}$ otherwise. It can be proved with the help of Lemma 3.83 given below that $\mathcal{A}^M$ in the C_i-mode accepts $L'(\mathcal{A})$. Hence $\mathsf{Reg}'_i \subseteq \mathsf{Reg}^{\mathsf{C}}_i$.

$\mathsf{Reg}^{\mathsf{C}}_i \subseteq \mathsf{Reg}'_i$ can be proved like so. Let $\mathcal{A} = (S, E_i(\Sigma), \delta, I, F)$ be a monoid automaton which, in the C_i-mode, accepts a binoid language L. A parenthesizing automaton $\mathcal{A}'$ such that $L'(\mathcal{A}') = L$ can be defined in the following way:

$$\begin{aligned}
&\mathcal{A}' = (H' \cup V', H', V', \delta', \gamma', I', F'), \text{ where}\\
&H' = \{s^H \mid s \in S\} \cup \{i_*, f_*\}, \quad V' = \{s^V \mid s \in S\}, \quad i_*, f_* \notin S,\\
&\delta' = \{(p^H, \sigma, q^H), (p^V, \sigma, q^V) \mid \sigma \in \Sigma, (p, \sigma, q) \in \delta\} \cup \{(i_*, \sigma, f_*) \mid \sigma \in L\},\\
&\gamma' = \{(p^H, \omega, q^V), (p^V, \omega, q^H) \mid \omega \in \Omega_i, (p, \omega, q) \in \delta\},\\
&I' = \{p^H \mid \exists i \in I : (i, \bullet, p) \in \delta\} \cup \{p^V \mid \exists i \in I : (i, \circ, p) \in \delta\} \cup \{i_*\},\\
&F' = \begin{cases} \{f^H, f^V \mid f \in F\} & \text{if } \varepsilon \notin L,\\ \{f^H, f^V \mid f \in F\} \cup \{i_*\} & \text{if } \varepsilon \in L. \end{cases}
\end{aligned}$$

One can make use of the lemma below to verify the correctness of the above construction.

Lemma 3.83 ([Ném07]) *Suppose that $\mathcal{A}$ is a PA with i pairs of parentheses, p and q are two states of $\mathcal{A}$, and $w \in \Sigma^*(\bullet, \circ)$. Then there is a direct run $\mathbf{r}$ of $\mathcal{A}$ from p to q with* $\mathrm{Label}(\mathbf{r}) = w$ *iff there is a transition sequence of $\mathcal{A}$ from p to q such that the middle components of the transitions give an i-cterm representation of w without the type sign.*

Now let us state some corollaries of Theorem 3.82.

Corollary 3.84 ([Ném07]) $\mathsf{Reg}^{\mathsf{FB}} = \mathsf{Reg}'_1$, *so* Reg'_1 *is closed under complementation.*

The next result shows that the more general class of recognizable binoid languages can also be captured by monoid mode acceptance.

Corollary 3.85 ([Ném07]) $\mathsf{Reg} = \mathsf{Reg}^{\mathsf{C}}_\infty = \mathsf{Reg}^{\mathsf{T}}_\infty$.

Proposition 2.21 and the concept of relativized regularity lead to the following result, whose proof is straightforward.

Corollary 3.86 ([Ném07]) *For all $i \geq 0$, any binoid language in Reg'_i has a deterministic context-free i-term (and i-cterm) representation. In particular, for any $L \in \mathsf{Reg}'_1$ the word languages L^{tm} and L^{ctm} are deterministic context-free.*

3.12.3 Conclusions

Now let us summarize certain key points of this section. The first is that Theorem 3.82 is effective in the sense that, for a PA, an equivalent monoid automaton (for the T_i-mode or C_i-mode) can be constructed and vice versa. Furthermore, the transition algorithm increases the number of states of the automaton by just a constant factor.

It seems that there are two ways of defining a recognizable binoid language L when using the monoid approach. The first is by taking an appropriate i-term or i-cterm representation of L, and defining it by a finite automaton or regular monoid expression. The second is by supplying L^{tm} or L^{ctm} directly via some visibly pushdown automaton [AM04] or deterministic pushdown automaton.

From a regularity point of view three classes of binoid languages might be of interest to us, namely $\mathsf{Reg}^{\mathsf{FM}}$, Reg'_1 and $\mathsf{Reg} = \mathsf{Rec}$, where the inclusion relations for them are $\mathsf{Reg}^{\mathsf{FM}} \subsetneq \mathsf{Reg}'_1 \subsetneq \mathsf{Reg}$. Note that, unlike the two other classes, $\mathsf{Reg}^{\mathsf{FM}}$ is not closed under complementation. Moreover, Theorems 3.32 and 3.82 tell us that in order to attain the general concept of recognizability (Rec) we need to handle several pairs of parenthesis symbols and unambiguous representations. However, this unambiguity can be avoided since from the proof of $\mathsf{Rec} \subseteq \mathsf{Reg}$ (Theorem 3.35) we obtain a PA that can be regarded as deterministic. This is not surprising at all, as recognizability is clearly a deterministic notion.

Another key point of our above results is that, by generalizing the idea of Hashiguchi et al., we managed to reduce the investigation of recognizable binoid languages to the classical theory of word languages. But the well-developed theory of monoid automata and word languages cannot be applied directly since the reduction has been done via the concept of relativized regularity. Hence it would be desirable to provide a detailed exposition of relativized regularity and find out how the methods of monoid automata (determinization, minimization and so on) can be transformed into automata over biwords, and, more generally, learn what effect the theory of word languages has on the theory of binoid languages. Finally, a deeper understanding of automata models, and especially the phenomenon of two-dimensional iterations, may also lead us to an operational characterization of the class of recognizable binoid languages.

Chapter 4

Parenthesizing Büchi-Automata

In this chapter we will now turn out thoughts to the issue of infinite biwords. Naturally our treatment will be based both on the concept of the finite case (Chapters 2 and 3) and on ideas found in the theory of infinite words (more precisely ω-words) [PP93, PP04, Wil94]. First, we will define ω-biwords and ω-binoid languages in analogy with ω-words and ω-languages. For this our starting point will be the infinite biposet representation of ω-biwords. (We will discuss representation by infinite terms later on.) The main goal of the chapter is to extend the validity of Theorems 3.35 and 3.70. Namely we would like to prove that – with an appropriate generalization of the concept of parenthesizing automata – the equivalence of recognizability, regularity and MSO-definability holds for ω-binoid languages as well.

4.1 ω-biwords and ω-bisemigroups

In order to introduce the concept of infinite biwords we first need operations with which they can be constructed from the finite biwords. Similarly to the classical case of ω-words this can be done in two ways: by an ω-product operation [PP04] or by an ω-power operation [Wil94]. In fact we need two ω-products or two ω-powers, both of which exist in a horizontal and a vertical form.

An ω-product operation constructs an infinite biword from an infinite countable number of finite biwords by writing them horizontally or vertically one after the other. In contrast to this, an ω-power operation takes a single finite biword and repeats it (horizontally or vertically) an infinite number of time. Therefore ω-products are of type $\mathsf{finite} \times \mathsf{finite} \times \ldots \to \mathsf{infinite}$, while the ω-powers are of type $\mathsf{finite} \to \mathsf{infinite}$. Apart form these operations we permit the formation of the horizontal/vertical product of a finite

biword with an infinite one. Obviously these operation are of type finite $\times$ infinite $\rightarrow$ infinite.

All the above operations can be defined in a straightforward way by using biposet representations of finite and infinite biwords. Hence in this subsection we will briefly summarize the main results of [ÉN05] on infinite biposets. First, we will introduce the operations that allows to construct infinite biposets from finite ones: the horizontal and vertical ω-products and the horizontal and vertical ω-powers.

Now suppose that $P_1, P_2, \ldots$ are pairwise disjoint finite biposets. Then their *horizontal ω-product* is defined as

$$\omega_\bullet(P_1, P_2, \ldots) := (P_1 \cup P_2 \cup \ldots, <_h, <_v, \lambda),$$

where

$$<_h := \bigcup_{i=1}^{\infty} <_h^{P_i} \ \cup \bigcup_{i<j} (P_i \times P_j), \qquad <_v := \bigcup_{i=1}^{\infty} <_v^{P_i}$$

and

$$\lambda := \lambda_{P_1} \cup \lambda_{P_2} \cup \ldots$$

The *vertical ω-product* $\omega_\circ(P_1, P_2, \ldots)$ is defined symmetrically. We will also write horizontal and vertical ω-products as $P_1 \bullet P_2 \bullet \ldots$ and $P_1 \circ P_2 \circ \ldots$, respectively. Moreover, the two ω-product operations naturally induce a *horizontal* and a *vertical power* operation, that is

$$P^{\omega\bullet} := P \bullet P \bullet P \bullet \ldots, \text{ and } P^{\omega\circ} := P \circ P \circ P \circ \ldots$$

Note that the definition of the product operations applies to both finite and infinite operands. However, in order to avoid constructing biposets which have chains not contained in ω, we will restrict the product operations $P \bullet Q$ and $P \circ Q$ just to a finite biposet P. Of course, the biposet Q may be finite or infinite, while the ω-product and ω-power operations are applied only to finite biposets. These restrictions seem to be necessary for the proofs presented later on.

All the restrictions we have just described imply that we should use two-sorted algebras as our algebraic framework to distinguish between finite and infinite elements. Using multi-sorted algebras is quite common for establishing an appropriate algebraic framework for various structures in computer science [Cou96]. Their application seems unavoidable also in the case of infinite word languages [PP04, Wil94]. Of course this will introduce a certain amount of technical complexity. But, fortunately, for biwords this can be carried out in complete analogy with the case of finite and infinite words (cf. [PP04]). But, as a minor difference from [PP04], we will assume the binary product

operations to be appropriately polymorphic, i.e. we will use the same notation for the product of two finite biword/biposets and for the product of a finite and an infinite biword/biposet.

Thus we will call an algebra $\mathcal{B} = (B_F, B_I, \bullet, \circ, \omega_\bullet, \omega_\circ)$ an *ω-bisemigroup* if it satisfies the following identities

$$\begin{aligned} x * (y * u) &= (x * y) * u, && (4.1)\\ x * \omega_*(x_1, x_2, \ldots) &= \omega_*(x, x_1, x_2, \ldots), && (4.2)\\ \omega_*(x_1 * \ldots * x_{k_1-1},\, x_{k_1} * \ldots * x_{k_2-1},\, \ldots) &= \omega_*(x_1, \ldots, x_{k_1-1},\, x_{k_1}, && (4.3)\\ & \qquad x_{k_1+1}, \ldots, x_{k_2-1},\, \ldots), \end{aligned}$$

for all $x, y, x_1, x_2, \ldots \in B_F$, $u \in B_F \cup B_I$, $* \in \{\bullet, \circ\}$, and for all increasing sequences of positive integers $k_1 < k_2 < \ldots$

A *homomorphism* of ω-bisemigroups $(C_F, C_I, \bullet, \circ, \omega_\bullet, \omega_\circ) \to (D_F, D_I, \bullet', \circ', \omega'_\bullet, \omega'_\circ)$ is a pair of functions $h = (h_F : C_F \to D_F, h_I : C_I \to D_I)$ that jointly preserve the operations. Congruences of ω-bisemigroups are defined in the usual way – they are such pairs of equivalence relations that are jointly respect the operations.

Remark 4.1 In this chapter we interested in languages which consist solely of infinite biwords. Hence for technical reasons it is preferable to work without the empty biword and without the empty biposet. This does not place any restriction on our theory, as the regular, recognizable and MSO-definable classes of ω-binoid languages (defined later) would be the same if we included the empty biword. Although one could define the notion of *ω-binoids*, (which would include an identity ε) as a three-sorted algebra. Besides the sorts of finite and infinite elements this algebra would also have a distinct sort for the identity and some additional equations

$$\begin{aligned} \varepsilon * \alpha &= \alpha * \varepsilon = \alpha \text{ and} && (4.4)\\ \varepsilon^{\omega_*} &= \varepsilon && (4.5) \end{aligned}$$

for all elements α and $* \in \{\bullet, \circ\}$.

Now let $\Sigma^\infty(\bullet, \circ)$ denote the *ω-bisemigroup generated freely by* Σ in the variety of all ω-bisemigroups. Needless to say the set of its finite elements is isomorphic to $\Sigma^+(\bullet, \circ)$, i.e. the free bisemigroup (generated by Σ). If we let $\Sigma^\omega(\bullet, \circ)$ denote the infinite elements of $\Sigma^\infty(\bullet, \circ)$, we have that

$$\Sigma^\infty(\bullet, \circ) := (\Sigma^+(\bullet, \circ), \Sigma^\omega(\bullet, \circ), \bullet, \circ, \omega_\bullet, \omega_\circ)$$

is an ω-bisemigroup.

The elements of $\Sigma^\omega(\bullet,\circ)$ will be called *ω-biwords*, while subsets of $\Sigma^\omega(\bullet,\circ)$ will be referred to as *ω-binoid languages*. In what follows we will study them.

We call a Σ-labeled biposet *constructible* if it can be generated from the singleton Σ-labeled biposets by the (restricted) binary product operations $\bullet$ and $\circ$, and by the ω-ary product operations $\omega_\bullet$ and $\omega_\circ$.

Recall from Section 2.6.2 that $\mathrm{SPB}(\Sigma)$ denotes the collection of biposets which can be generated from the singletons corresponding to the letters in Σ by the two product operations $\bullet$ and $\circ$. Hence $\mathrm{SPB}(\Sigma)$ does not contain the empty biposet ε. Also the biposets in $\mathrm{SPB}(\Sigma)$ are called series-parallel biposets or just sp-biposets. Note that $\mathrm{SPB}(\Sigma)$ is simply the set of those constructible biposets which are finite. Next, let $\mathrm{ISPB}(\Sigma)$ denote the set of infinite constructible biposets, and let

$$\omega\mathrm{SPB}(\Sigma) := (\mathrm{SPB}(\Sigma), \mathrm{ISPB}(\Sigma), \bullet, \circ, \omega_\bullet, \omega_\circ)$$

stand for the two-sorted algebra of all constructible biposets over Σ.

It is clear that $\omega\mathrm{SPB}(\Sigma)$ satisfies (4.1)–(4.3), hence it is an ω-bisemigroup. Now it can be seen that the set of *all* finite and countably infinite biposets also form an ω-bisemigroup, and $\omega\mathrm{SPB}(\Sigma)$ is the smallest subalgebra of this ω-bisemigroup that contains Σ. The infinite counterpart of the free algebra theorem for sp-biposets (Theorem 2.14) is the following.

Theorem 4.2 ([ÉN05]) *The algebra $\omega\mathrm{SPB}(\Sigma)$ is freely generated by Σ in the variety of ω-bisemigroups, i.e. $\omega\mathrm{SPB}(\Sigma)$ is isomorphic to $\Sigma^\infty(\bullet,\circ)$.*

Since the proofs required for this and others below are long and their details are not really necessary to understand the concepts and results for parenthesizing Büchi-automata and regular ω-binoid languages, we will omit them here. The interested reader can find all the relevant details in [ÉN05].

A graph-theoretic characterization of infinite constructible biposets is also given in [ÉN05]. This, of course, is a suitable generalization of the "generalized N-free" condition of the finite case (Corollary 2.27).

Theorem 4.3 ([ÉN05]) *An infinite biposet $(P, <_h, <_v, \lambda)$ is in $\mathrm{ISPB}(\Sigma)$ if and only if P is complete, and both posets $(P, <_h)$ and $(P, <_v)$*

(i) are N-free,

(ii) are free of "upward combs",

(iii) are free of "downward combs", and

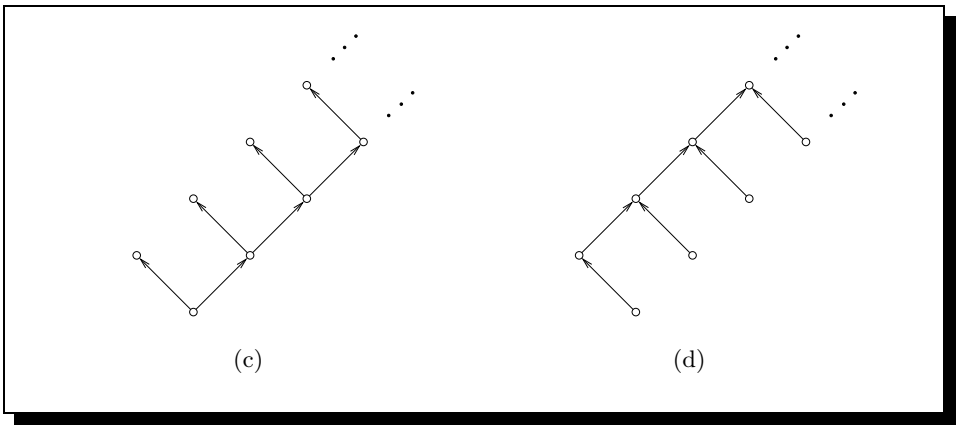

Figure 4.1: *An upward comb (a) and a downward comb (b).*

(iv) all of their principal ideals[1] *are finite.*

where "upward comb" and "downward comb" are infinite posets. These are depicted in Figure 4.1[2]*.*

In order to simplify the notations, in the following we will use $*$ to indicate any of the $\bullet$ and $\circ$ operations. Sometimes we also give subscripts to the $*$-s, but in any formula all $*$ symbols, without subscript or with the same subscript, will always denote the same operation.

A decomposition of an ω-biword w into an ω-product of infinitely many finite biwords $w = w_1 * w_2 * \dots$ is said to be *maximal* if every w_i is $*$-irreducible. If w is an ω-biword, we say that w is *primitive* if it can be written as $w_1 * w_2 * \dots$ for some finite biwords $w_1, w_2, \dots$ Now each ω-biword can be generated from the primitive biwords by multiplication with finite biwords from the left. We will define the *rank* of an ω-biword w as the least number of left multiplications with finite biwords needed to construct w from the primitive ω-biwords. The rank of w will be denoted by $\mathrm{Rank}(w)$.

It is not hard to prove that if an ω-biword w is not primitive, then it can be uniquely written as $w = w' * w''$, where w'' is $*$-irreducible and $\mathrm{Rank}(w'') < \mathrm{Rank}(w)$. A direct consequence of this fact is that every ω-biword can be written in the form

$$w_1 *_1 (w_2 *_2 (w_3 *_3 \dots w_k *_k (u_1 *_{k+1} u_2 *_{k+1} u_3 *_{k+1} \dots))), \tag{4.6}$$

where all w_i and u_i are finite biwords in $\Sigma^+(\bullet, \circ)$, and $*_1, *_2, *_3 \dots$ is an alternating sequence of the $\bullet$ and $\circ$ operations. Moreover, this form is unique provided that every

[1] A principle ideal is a collection of all elements which are less than or equal to a given element.
[2] See [ÉN05] for precise definitions.

u_i is $*_{k+1}$-irreducible. In this case we call it the *normal form* of w. Note that if (4.6) is the normal form of w, then $\mathrm{Type}(w) = *_1$ and $\mathrm{Rank}(w) = k$.

4.2 Tree and Term Representations of ω-biwords

Here we will outline the changes (from the finite case) that should be made if one intends to represent ω-biwords by infinite terms and trees. Let w be an ω-biword. As in the finite case, we will denote the biposet, tree and term representations of w, by w^{bp}, w^{tr} and w^{tm}, respectively.

The only thing we really need to describe is how to handle ω-products as $w = w_1 \bullet w_2 \bullet \ldots$ The definition of w^{tr} is straightforward if we allow ω-branching in trees. The tree w^{tr} has a root labeled by $\bullet$, and this root has ω branches which connects the tree representations of all the horizontally irreducible components of the w_i-s ($i \geq 1$).

Also, only slight changes need to be made in the term representation. First of all we require an additional symbol of parenthesis: '['. Now the term representations of an ω-biword in $\Sigma^\omega(\bullet, \circ)$ is an ω-word over the extended alphabet

$$E'(\Sigma) := \Sigma \cup \{\, \langle\,,\, \rangle,\, [\,,\, \bullet\,,\, \circ \,\}.$$

The definitions of the horizontal and vertical forms can also be extended in an appropriate way. In the representation of a product of a finite biword with an infinite one, we use the [symbol if the type of the product differs from the type of the infinite factor. Thus we should modify Definition 2.15, as follows.

Definition 4.4 *If $w \in \Sigma^\omega(\bullet, \circ)$, let w^{tm}, the* term representation *of w, be an ω-word over $E'(\Sigma)$, defined by induction on* $\mathrm{Rank}(w)$ *as follows.*

(i) *If* $\mathrm{Rank}(w) = 0$ *and $w = w_1 \bullet w_2 \bullet \ldots$ is a primitive horizontal ω-biword with $w_1, w_2, \ldots \in \Sigma^+(\bullet, \circ)$, then $w^{tm} := \mathrm{Hform}(w_1) \bullet \mathrm{Hform}(w_2) \bullet \ldots$*

(ii) *If* $\mathrm{Rank}(w) = 0$ *and $w = w_1 \circ w_2 \circ \ldots$ is a primitive vertical ω-biword with $w_1, w_2, \ldots \in \Sigma^+(\bullet, \circ)$, then $w^{tm} := \mathrm{Vform}(w_1) \circ \mathrm{Vform}(w_2) \circ \ldots$*

(iii) *If* $\mathrm{Rank}(w) > 0$ *and $w = w_1 \bullet w_2$ is a horizontal ω-biword with $w_1 \in \Sigma^+(\bullet, \circ)$, $w_2 \in \Sigma^\omega(\bullet, \circ)$ and* $\mathrm{Rank}(w_2) < \mathrm{Rank}(w)$*, then $w^{tm} := \mathrm{Hform}(w_1) \bullet \mathrm{Hform}'(w_2)$.*

(iv) *If* $\mathrm{Rank}(w) > 0$ *and $w = w_1 \circ w_2$ is a vertical ω-biword with $w_1 \in \Sigma^+(\bullet, \circ)$, $w_2 \in \Sigma^\omega(\bullet, \circ)$ and* $\mathrm{Rank}(w_2) < \mathrm{Rank}(w)$*, then $w^{tm} := \mathrm{Vform}(w_1) \circ \mathrm{Vform}'(w_2)$.*

In (iii), $\mathrm{Hform}'(w)$ *denotes the* horizontal form *of the ω-biword w, defined as follows:*

$$\mathrm{Hform}'(w) \;:=\; \begin{cases} w^{tm} & \text{if } w \text{ is a horizontal } \omega\text{-biposet,} \\ [\, w^{tm} & \text{if } w \text{ is a vertical } \omega\text{-biposet.} \end{cases}$$

In (iv), $\mathrm{Vform}'(w)$, *the* vertical form *of the ω-biword w, is defined symmetrically.*

An example of an ω-binoid language and the term representation of its elements are given in the section below.

4.3 Recognizability of ω-binoid Languages

Recall that a language consisting of finite nonempty biwords is said to be *recognizable* if it is recognized by a homomorphism into a finite bisemigroup, i.e. $L \subseteq \Sigma^+(\bullet,\circ)$ is recognizable if and only if $L = \varphi^{-1}(F)$, for some bisemigroup homomorphism $\varphi : \Sigma^+(\bullet,\circ) \to B$, where B is a finite bisemigroup, and $F \subseteq B$.

Similarly, for a language that contains both finite biwords and ω-biwords, $L = (L_F, L_I) \subseteq \Sigma^\infty(\bullet,\circ)$, is recognizable if and only if there is a finite ω-bisemigroup $\mathcal{B} = (B_F, B_I)$, a subset of it, $T = (T_F, T_I) \subseteq (B_F, F_I)$, and a homomorphism $\varphi = (\varphi_F, \varphi_I) : \Sigma^\infty(\bullet,\circ) \to \mathcal{B}$ such that $L = \varphi^{-1}(T)$. Here $(T_F, T_I) \subseteq (B_F, F_I)$ means $T_F \subseteq B_F$ and $T_I \subseteq F_I$; moreover, $L = \varphi^{-1}(T)$ stands for $L_F = \varphi_F^{-1}(T_F)$ and $L_I = \varphi_I^{-1}(T_I)$.

Example 4.5 Let $\Sigma = \{a, b, c\}$, and consider the following language $L \subseteq \Sigma^\infty(\bullet,\circ)$ of ω-biwords

$$L = \{\, c^{\omega\bullet},\ a \bullet (b \circ (c^{\omega\bullet})),\ a \bullet (b \circ (a \bullet (b \circ (c^{\omega\bullet})))), \dots \}.$$

I.e. L is the least solution of the fixed point equation $a \bullet (b \circ X) + c^{\omega\bullet} = X$. It is not hard to show that L is recognizable. Indeed, consider the finite bisemigroup $B = (B_F, B_I)$, where $B_F = \{\, d_a, d_b, d_c, 0 \,\}$, and $B_I = \{\, t_1, t_2, t_3, \underline{0} \,\}$. The binary product operations are given by $d_c \bullet d_c = d_c$, and all other binary products of two finite elements are equal to 0; moreover, $d_c \bullet t_1 = t_1$, $d_b \circ t_1 = t_2$, $d_a \bullet t_2 = t_3$, $d_b \circ t_3 = t_2$, and all other products of a finite element with an infinite one are equal to $\underline{0}$. Next, the ω-product operations are given by $d_c^{\omega\bullet} = t_1$, and all other ω-products are equal to $\underline{0}$. Now if we take the homomorphism $\varphi : \omega\mathrm{SPB}(\Sigma) \to \mathcal{B}$ that is induced by the mapping $a \mapsto d_a$, $b \mapsto d_b$, $c \mapsto d_c$, then $L = \varphi_I^{-1}(\{\, t_1, t_3 \,\})$. This shows that L is recognizable.

4.4 Logical Definability of ω-binoid Languages

In this section we will extend the concept of MSO-definable binoid languages (introduced in Section 3.11) to ω-binoid languages.

The syntax of MSO-formulas will remain the same as for finite biwords. Thus the construction of the formulas can be described as follows:

$$\varphi := Q_a(x) \mid x <_h y \mid x <_v y \mid X(x) \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \to \varphi \mid \varphi \equiv \varphi \mid \exists x\varphi \mid \forall x\varphi \mid \exists X\varphi \mid \forall X\varphi,$$

where $a \in \Sigma$, x is a first-order variable and X is a second-order variable.

The interpretation of the formulas also remain the same, of course we use the biposet representation of biwords as in the finite case. But now we will also interpret our formulas over infinite constructible biposets. Hence the value of a second order variable X can be both a finite and an infinite set of positions. The following simple proposition can be proved easily by induction on the construction of P.

Proposition 4.6 *If $P = (P, <_h, <_v, \lambda)$ is a (finite or infinite) constructible biposet, then $<_h \cup <_v$ is a linear order on P.*

With Proposition 4.6 we can express the finiteness of a biposet as follows:

$$\varphi_{\text{fin}} := \exists x \forall y [\, (y <_h x) \vee (y <_v x) \vee (x = y) \,]. \tag{4.7}$$

Here, of course, $x = y$ can be defined as $\neg(x <_h y) \wedge \neg(x <_v y) \wedge \neg(y <_h x) \wedge \neg(y <_v x)$. Now we can assign an ω-binoid language to any MSO-formula:

$$L_{\omega,\varphi} := \{\, w \in \Sigma^\omega(\bullet, \circ) \mid w^{bp} \models \varphi \,\}.$$

Notice that because of (4.7)

$$L_{\omega,\varphi} = \{\, w \in \Sigma^\infty(\bullet, \circ) \mid w^{bp} \models \varphi \wedge \neg\varphi_{\text{fin}} \,\}.$$

Definition 4.7 *We say that an ω-binoid language $L \subseteq \Sigma^\omega(\bullet, \circ)$ is* MSO-definable *if there is a sentence φ with $L = L_{\omega,\varphi}$. Next, let* MSO^ω *denote the class of all* MSO-*definable ω-binoid languages.*

4.5 Regularity of ω-binoid Languages

Our next task is to define parenthesizing Büchi-automata so that a language is recognizable if and only if it can be accepted by this kind of automaton. A straightforward approach would be (as in the case of infinite words) to use the same accepting device and just extend the notion of run appropriately for the acceptance of ω-biwords, but as we shall see (in Remark 4.13) this cannot be accomplished. Therefore our definition will be the following.

Definition 4.8 ([Ném06]) *A* parenthesizing Büchi-automaton *is a tuple* $\mathcal{A} := (S, H, V, \Sigma, \Omega, [, \delta, \beta, \gamma, I, F)$*, where* $\mathcal{A}' := (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ *is a parenthesizing automaton, called the* underlying parenthesizing automaton *of* $\mathcal{A}$*. And the new components are the following:*

- $[\notin (\Sigma \cup \Omega)$ *is the* separating parenthesis*, and*

- $\beta \subseteq (H \times \{ [\} \times V) \cup (V \times \{ [\} \times H)$ *is the* separating transition relation.

For the sake of simplicity we shall write $[p, w, q]_{\mathcal{A}}$ instead of $[p, w, q]_{\mathcal{A}'}$ if w is a finite biword, and $\mathcal{A}'$ is the underlying parenthesizing automaton of the parenthesizing Büchi-automaton $\mathcal{A}$.

Remark 4.9 As we have seen in Theorem 3.32, if we wish to accept all regular binoid languages, we cannot give a universal upper bound for the number of parentheses used in parenthesizing automata. On the other hand, as we need not close parentheses of the separating transitions, a single symbol by itself is enough to change the type of the state at the borders of a "finite − infinite" product.

Next, we will define when a parenthesizing automaton $\mathcal{A}$ accepts an ω-biword w from a given state p. For this, we choose Büchi's approach: for acceptance a run must contain a final state r (in certain "outer" positions) infinitely many times. Let $[p, w, r]^{\infty}_{\mathcal{A}}$ denote this fact. Its definition distinguishes two cases and uses induction on the rank of w. Recall that we write $\mathrm{Type}(p) = \bullet$ when p is a horizontal state, and $\mathrm{Type}(p) = \circ$ when p is a vertical state of $\mathcal{A}$. Similarly, $\mathrm{Type}(w) = \bullet$ ($\mathrm{Type}(w) = \circ$) indicates that w is a horizontal (vertical, resp.) biword.

Definition 4.10 ([Ném06]) *Suppose that* $\mathcal{A} = (S, H, V, \Sigma, \Omega, [, \delta, \gamma, \beta, I, F)$ *is a parenthesizing Büchi-automaton,* p *and* r *are in* S*, and* w *is an* ω*-biword in* $\Sigma^{\omega}(\bullet, \circ)$*. We write* $[p, w, r]^{\infty}_{\mathcal{A}}$ *in the following cases.*

i) $\mathrm{Type}(p) = \mathrm{Type}(w)$*, and either*

  α) w can be written as $w = w_0 * w_1 * w_2 * \ldots$*, where each* w_i *is a finite (not necessarily* $*$*-irreducible) biword such that* $[p, w_0, r]_{\mathcal{A}}$ *and* $[r, w_i, r]_{\mathcal{A}}$ *for* $i > 0$*; or*

  β) $w = w' * w''$*, where* $w' \in \Sigma^{+}(\bullet, \circ)$*,* $w'' \in \Sigma^{\omega}(\bullet, \circ)$*,* $\mathrm{Rank}(w'') < \mathrm{Rank}(w)$*, and there is a state* $q \in S$ *such that* $[p, w', q]_{\mathcal{A}}$ *and* $[q, w'', r]^{\infty}_{\mathcal{A}}$*, the latter being defined inductively.*

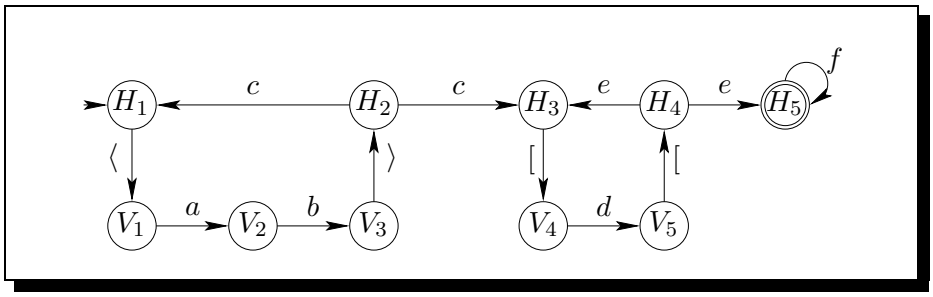

Figure 4.2: *A parenthesizing Büchi-automaton.*

ii) $\mathrm{Type}(p) \neq \mathrm{Type}(w)$*, and there exists a state* $p' \in S$ *such that* $\mathcal{A}$ *has a separating transition* $(p, [, p') \in \beta$*, and* $[p', w, r]^{\infty}_{\mathcal{A}}$ *holds according to case i) above.*

Definition 4.11 *A parenthesizing Büchi-automaton* $\mathcal{A}$ *accepts the following* ω*-binoid language*

$$L_\omega(\mathcal{A}) := \{\, w \in \Sigma^\omega(\bullet, \circ) \mid [i, w, f]^{\infty}_{\mathcal{A}} \text{ for some } i \in I \text{ and } f \in F \,\}.$$

As before, a language $L \subseteq \Sigma^\omega(\bullet, \circ)$ *is* regular *if there is a parenthesizing Büchi-automaton* $\mathcal{A}$ *such that* $L = L_\omega(\mathcal{A})$*. Next, let* Reg^ω *denote the class of regular* ω*-binoid languages.*

Similarly to Definition 3.7, one could also define infinite runs in a formal way. In this case, runs are ω-words over the union of the sets of labeling, parenthesizing and separating transitions. Later we will employ the same notations as in the finite case: $\mathrm{Runs}(\mathcal{A})$, $\mathrm{Biposet}(\mathbf{r})$ and so on.

Example 4.12 Figure 4.2 shows a parenthesizing Büchi-automaton. The horizontal states are those labeled by H_i while the vertical states are those labeled by V_j for some i and j. There is a single initial state H_1 and a single final state H_5. The angle brackets indicate parenthesizing transitions, while the square brackets represent separating transitions. It is easy to verify that this automaton only accepts ω-biwords of the form

$$(a \circ b) \bullet c \bullet (a \circ b) \bullet c \bullet \ldots \bullet (a \circ b) \bullet c \bullet (d \circ (e \bullet (d \circ (e \bullet \ldots (d \circ (e \bullet f \bullet f \bullet f \ldots)) \ldots))))$$

Notice that the term representation of the ω-biword above is

$$\langle a \circ b \rangle \bullet c \bullet \langle a \circ b \rangle \bullet c \bullet \ldots \bullet \langle a \circ b \rangle \bullet c \bullet [d \circ [e \bullet [d \circ [e \bullet \ldots [d \circ [e \bullet f \bullet f \bullet f \ldots$$

Remark 4.13 As we mentioned earlier, we cannot use the original model of parenthesizing automata for the acceptance of ω-biwords. First, there is no meaningful definition of closing a parenthesis after processing an infinite subbiword. E.g. suppose that $w = u \bullet v$ is an ω-biword, where u is a finite biword (either horizontal or vertical), and v is a vertical ω-biword. Now, if

there is a finite run $[p, u, q]_{\mathcal{A}}$ for some horizontal states p and q, we must open a parenthesis by some transition $(q, \langle, r)$ in order to arrive at a vertical state r, from where the acceptance of the vertical ω-biword v can be started. However, as v is infinite, there is no possibility of closing this parenthesis. Second, it can be demonstrated that we must distinguish between the normal and these "non-closable" parenthesizing transitions, otherwise there would be recognizable ω-binoid languages that could not be accepted by Büchi-automata. To see this, consider the language $L := \{a \bullet (b \circ c) \bullet d^{\omega\bullet}, a \bullet (b \circ (a \bullet (b \circ c))) \bullet d^{\omega\bullet}, a \bullet (b \circ (a \bullet (b \circ (a \bullet (b \circ c))))) \bullet d^{\omega\bullet}, \ldots\}$. To accept L any automaton must remember to close all the parentheses that it opens.

4.6 From Regularity to Recognizability

As in the finite case (Theorem 3.35), the fact that regularity implies recognizability easily follows from the finite-state property of automata.

Theorem 4.14 ([Ném06]) *Every regular ω-binoid language is recognizable. Thus* $\mathsf{Reg}^{\omega} \subseteq \mathsf{Rec}^{\omega}$.

Proof. Let $L \subseteq \Sigma^{\omega}(\bullet, \circ)$ denote a regular ω-binoid language. We show how to transform a parenthesizing Büchi-automaton $\mathcal{A} = (S, H, V, \Sigma, \Omega, [, \delta, \gamma, \beta, I, F)$ accepting L into a finite ω-bisemigroup that recognizes L. The proof is analogous to the proof for ω-words [PP04].

Recall that $[p, w, q]_{\mathcal{A}}$ means that the underlying parenthesizing automaton of the parenthesizing Büchi-automaton $\mathcal{A}$ has a run on the finite biword w from state p to state q. Now, suppose that $\mathrm{Type}(p) = \mathrm{Type}(q) = *$, and consider a finite biword w. If w is $*$-irreducible, then take $w_1 = w$ and $m = 1$, otherwise let the maximal $*$-decomposition of w be $w = w_1 * w_2 * \ldots * w_m$ for some $m \geq 2$.

Now, according to Definition 3.3, there are states $p_0 = p, p_1, \ldots, p_m = q$ of type $*$ such that $[p_0, w_1, p_1]_{\mathcal{A}}, [p_1, w_2, p_2]_{\mathcal{A}}, \ldots, [p_{m-1}, w_m, p_m]_{\mathcal{A}}$ hold. Let us write $[p, w, q]_{\mathcal{A}+}$ to indicate the existence of such states for which $\{p_0, \ldots, p_m\} \cap F \neq \emptyset$. Thus $[p, w, q]_{\mathcal{A}+}$ if and only if there is at least one final state among the "outer" states of a possible run between p and q on w.

Next, we will define for any $w, u \in \Sigma^{+}(\bullet, \circ)$ and $w', u' \in \Sigma^{\omega}(\bullet, \circ)$

$$w \sim_F u \text{ iff } \forall\, p, q \in S : \big([p, w, q]_{\mathcal{A}} \Leftrightarrow [p, u, q]_{\mathcal{A}} \text{ and } [p, w, q]_{\mathcal{A}+} \Leftrightarrow [p, u, q]_{\mathcal{A}+}\big),$$
$$w' \sim_I u' \text{ iff } \forall\, p \in S : \big(\exists\, r \in F :\ [p, w', r]^{\infty}_{\mathcal{A}} \Leftrightarrow \exists\, r' \in F :\ [p, u', r']^{\infty}_{\mathcal{A}}\big).$$

Here one can verify that $\sim_F$ and $\sim_I$ are equivalence relations with finitely many equivalence classes. Furthermore, they satisfy the following equalities. If $w_i, u_i \in \Sigma^{+}(\bullet, \circ)$,

for $i = 1, 2, \ldots$, and $w', u' \in \Sigma^\omega(\bullet, \circ)$, $* \in \{\bullet, \circ\}$, then

$$\begin{array}{lcl} w_1 \sim_F u_1,\ w_2 \sim_F u_2 & \Rightarrow & w_1 * w_2 \sim_F u_1 * u_2, \\ w_i \sim_F u_i \text{ for } i > 0 & \Rightarrow & w_1 * w_2 * \ldots \sim_I u_1 * u_2 * \ldots, \text{ and} \\ w_1 \sim_F u_1,\ w' \sim_I u' & \Rightarrow & w_1 * w' \sim_I u_1 * u'. \end{array}$$

Hence the pair $(\sim_F, \sim_I)$ is in fact a congruence of ω-bisemigroups. Thus the quotient of $\Sigma^\infty(\bullet, \circ)$ / $(\sim_F, \sim_I)$ can be equipped with the structure of an ω-bisemigroup. Moreover, the canonical epimorphism of $\Sigma^\infty(\bullet, \circ)$ onto $\Sigma^\infty(\bullet, \circ)$ / $(\sim_F, \sim_I)$ recognizes $L(\mathcal{A})$. □

4.7 From Recognizability to Regularity

Below we intend to prove the converse of Theorem 4.14, namely that every recognizable ω-binoid is regular, i.e. each can be accepted by a parenthesizing Büchi-automaton.

Theorem 4.15 ([Ném06]) *Every recognizable ω-binoid language is regular. Thus* $\mathsf{Rec}^\omega \subseteq \mathsf{Reg}^\omega$.

Proof. Suppose that an ω-binoid language $L \subseteq \Sigma^\omega(\bullet, \circ)$ is recognized by a homomorphism $\varphi : \Sigma^\infty(\bullet, \circ) \to \mathcal{B}$, where $\mathcal{B} = (B_F, B_I)$ is a finite ω-bisemigroup, and $L = \varphi^{-1}(F)$ for some $F \subseteq B_I$.

Let us call an element $e \in B_F$ *horizontally idempotent* if it is idempotent with respect to the horizontal product, i.e. $e \bullet e = e$. Similarly, e is said to be *vertically idempotent* if $e \circ e = e$. This notion is important because every primitive ω-biword $w_0 * w_1 * \ldots$ can be written in the form $w_0' * w_1' * \ldots$, where $\varphi(w_0') = b$ and $\varphi(w_i') = e$ for all $i > 0$, where e is a $*$-idempotent element of B_F. This follows from an application of the Ramsey-theorem (cf. [PP04]). We can even assume that $b = b * e$, but we need not do this now.

Thus if we omit w_0' from the above ω-biword, then the remaining primitive ω-biword is $w_1' * w_2' * \ldots$, where $\varphi(w_i') = e$ for all $i > 0$. Let us call ω-biwords that can be written in such a form *e-$*$-primitive*.

For a given e and $*$, the set of all e-$*$-primitive ω-biwords can be easily accepted by a parenthesizing Büchi-automaton $\mathcal{A}_{e,*}$ constructed as follows. Since $\varphi^{-1}(e)$ is a recognizable set of finite biwords, then, by Theorem 3.35, it is also regular. So there is a parenthesizing (finite) automaton $\mathcal{A}$ which accepts $\varphi^{-1}(e)$. Moreover, by Lemma 3.23, it can be assumed that $\mathcal{A}$ is in $*$-normal form (i.e. in horizontal normal form if $* = \bullet$, or in vertical normal form if $* = \circ$). Hence $\mathcal{A}$ has a single initial state i and a single finite

state f, both of which are of type $*$. We can transform $\mathcal{A}$ into $\mathcal{A}_{e,*}$ just by merging i and f. We will refer to this fused state as the *basic state* of $\mathcal{A}_{e,*}$. Now it is obvious that if we regard $\mathcal{A}_{e,*}$ as a Büchi-automaton with its basic state as a single initial and final state, it will accept exactly the e-$*$-primitive ω-biwords.

Recall that the normal form of an ω-biword is

$$w_1 *_1 (w_2 *_2 \ldots (w_k *_k (u_1 *_{k+1} u_2 *_{k+1} \ldots))),$$

where $w_1, \ldots, w_k, u_1, u_2, \ldots$ are in $\Sigma^+(\bullet, \circ)$. We can assume that except for a finite factor u', the ω-biword $u_1 *_{k+1} u_2 *_{k+1} \ldots$ is e-$*_{k+1}$-primitive for some $*_{k+1}$-idempotent e. Thus we only need to build our automaton in such a way that it can also process the finite "introductory slice" $w_1 *_1 (w_2 *_2 (\ldots w_k *_k (u' *_{k+1}$ before the e-$*_{k+1}$-primitive tail.

Now assume that $B_I = \{t_1, t_2, \ldots, t_m\}$. We start to construct a Büchi-automaton $\mathcal{A}$ from the horizontal states H_0, vertical states V_0, with separating transitions β, where

$$\begin{aligned} H_0 &:= \{ t_1^\bullet, t_2^\bullet, \ldots, t_m^\bullet \}, \\ V_0 &:= \{ t_1^\circ, t_2^\circ, \ldots, t_m^\circ \}, \text{ and} \\ \beta &:= \{ (t_i^\bullet, [, t_i^\circ), (t_i^\circ, [, t_i^\bullet) \mid i = 1, \ldots, m \}. \end{aligned}$$

For all $b \in B_F$, there is a parenthesizing (finite) automaton $\mathcal{A}_b$ recognizing $\varphi^{-1}(b)$. We will incorporate these finite automata into $\mathcal{A}$.

More precisely, if p and q are states of $\mathcal{A}$ of the same type, say $*$, then one can take a copy of $\mathcal{A}_b$ in $*$-normal form and merge its initial state i and final state f with the states p and q of $\mathcal{A}$, respectively. In the following, we shall refer to this construction as an *extension* of $\mathcal{A}$ (between p and q) by $\varphi^{-1}(b)$. Let us denote it by

$$p \overset{\varphi^{-1}(b)}{\Longrightarrow} q.$$

We need to add the following extensions to $\mathcal{A}$:

$$t_i^* \overset{\varphi^{-1}(b)}{\Longrightarrow} t_j^* \quad \text{for all } t_i = b * t_j, \quad b \in B_F,\ t_i, t_j \in B_I, * \in \{ \bullet, \circ \}.$$

Now we obviously have

$$[\, t_i^*, w, t_j^* \,]_{\mathcal{A}} \Leftrightarrow t_i = \varphi(w) * t_j \text{ for any } w \in \Sigma^+(\bullet, \circ).$$

Furthermore,

$$[\, t_i^*, w_1 *_1 (w_2 *_2 (\ldots w_{k-1} * (w_k *_k, t_j^* \,]_{\mathcal{A}} \Leftrightarrow t_i = \varphi(w_1) *_1 (\varphi(w_2) *_2 (\ldots \varphi(w_k) *_k t_j)),$$

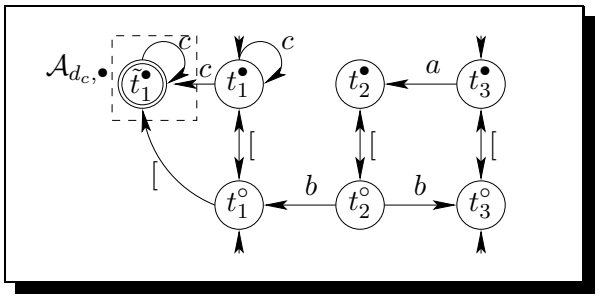

Figure 4.3: *A Büchi-automaton accepting the recognizable language of Example 4.5.*

where the left hand side is an abbreviation for the first part of an infinite run of $\mathcal{A}$ (as a Büchi-automaton) on any ω-biword beginning with $w_1 *_1 (w_2 *_2 \ldots (w_k *_k$.

Next, add an instance of $\mathcal{A}_{e,*}$ for each $*$-idempotent element e in B_F to $\mathcal{A}$, and ensure the reachability of the new components by adding some new transitions. In more detail, for $\mathcal{A}_{e,*}$, consider $t := e^{\omega *}$ and duplicate each transition arriving at t^* using the same source and label, but with the target of the basic state of $\mathcal{A}_{e,*}$ instead of t^*.

Our last task is to determine the initial and the final states. Let the initial states be the states $t^\bullet$ and t° for each $t \in F$, and set the basic states of the components $\mathcal{A}_{e,*}$-s as final states.

Next, it can be argued by induction on the rank of the ω-biwords that this automaton in fact accepts $L = \varphi^{-1}(F)$. We will omit the formal proof of this. □

Example 4.16 Figure 4.3 shows a parenthesizing Büchi-automaton that was constructed according to the proof of Theorem 4.15 from the homomorphism φ, the ω-bisemigroup B, and the set $F \subseteq B$ of Example 4.5. We omitted the two shrink states $\underline{0}^\bullet, \underline{0}^\circ$ that correspond to the infinite zero element, and also the transitions pointing to them. We should mention, however, that this example represents a somewhat special case, since, for every $x \in B_F$ the extension by $\varphi^{-1}(x)$ is a single transition, and there is only one idempotent in B_F. Of course, in the general case the automaton we construct can have a more complex structure.

4.8 From Regularity to MSO-definability

In this section we will prove the equivalence of regularity and MSO-definability. First of all, it is not hard to demonstrate that MSO-definability implies recognizability, and hence regularity. This can be proved via formula induction using the closure properties of the recognizable sets – more precisely, the closure under Boolean operations and

direct letter-to-letter homomorphisms. See Chapter III.1 of Straubing [Str94] for a similar argument. Thus we have the following theorem.

Theorem 4.17 ([Ném06]) *Every* MSO*-definable ω-binoid language is regular. Thus* $\mathsf{MSO}^\omega \subseteq \mathsf{Reg}^\omega$.

In the rest of this chapter we will attempt to prove the converse of this which can also be stated as follows:

Theorem 4.18 ([Ném06]) *Every regular ω-binoid language is* MSO*-definable. Thus* $\mathsf{Reg}^\omega \subseteq \mathsf{MSO}^\omega$.

Since the proof involves the construction of an MSO-formula for a given regular ω-binoid language, here we will represent biwords (resp. ω-biwords) by sp-biposets (resp. constructible infinite biposets). But before the proof, let us introduce a couple of definitions and lemmas.

Recall that if w is a biword or an ω-biword, then w^{bp} denotes its biposet representation, w^{tr} denotes its tree representation and w^{tm} stands for its term representation. Similarly, if P is a constructible biposet, then there is a unique biword or ω-biword w such that $P = w^{bp}$. In this case let $P^{tr} := w^{tr}$ and $P^{tm} = w^{tm}$. It is obvious that for any leaf node in P^{tr} there is a corresponding vertex in P. Hence, we may and shall identify the leaves of P^{tr} with the corresponding vertices of P. This allows us to speak about elements and subsets of P as those of P^{tr}. Similarly, we can identify vertices of P with the corresponding letters in the term representation P^{tm}.

The notion of clans is a useful idea whose definition comes from the theory of 2-structures [ER90] and texts [ER93, HtP97]. If $(P, <_h, <_v, \lambda)$ is a finite or infinite constructible sp-biposet, a subset X of P is said to be a *clan* of P if for all $x, y \in X$, $z \in P \setminus X$ and for each relation $\rho \in \{<_h, <_v, <_h^{-1}, <_v^{-1}\}$

$$x \rho z \Leftrightarrow y \rho z.$$

Two clans X and Y *overlap* if $X \cap Y \neq \emptyset$, $X \setminus Y \neq \emptyset$ and $Y \setminus X \neq \emptyset$. A clan is called a *prime clan* if it does not overlap with any other clan. A clan of P is called a *proper clan* if it is neither a singleton nor equal to P.

Example 4.19 In the sp-biposet of Example 2.18, the clans of P are the following: the singletons, P, $\{1,2,3,4\}$, $\{2,3,4\}$, $\{3,4\}$, $\{2,3,4,5,6\}$ and $\{5,6\}$. Since only $\{1,2,3,4\}$ and $\{2,3,4,5,6\}$ overlap, the other clans are prime clans as well. Thus the proper prime clans are $\{2,3,4\}$, $\{3,4\}$ and $\{5,6\}$. As we will see later in Lemma 4.22, these are the sets which are surrounded by parentheses in the term representation of P.

The proof of the following lemma is trivial and is left to the reader.

Lemma 4.20 ([Ném06]) *The property of being a clan, a prime clan or a proper prime clan can be expressed in* MSO *logic.*

Recall that by Proposition 4.6 $<_h \cup <_v$ is a linear order on any finite or infinite constructible biposet $(P, <_h, <_v, \lambda)$. In the following let $<$ denote the $<_h \cup <_v$ relation, and we will interpret the functions $+$ and $-$ according to this linear order as well.

By definition, clans form sections (or intervals) with respect to $<$. That is, if X is a clan then $x \in X$, $y \in X$ and $x < z < y$ imply $z \in X$. Hence, we can talk about *prefix* and *suffix* relations among the clans of P. Formally,

$$\begin{aligned} \mathrm{Prefix}(X,Y) &:= X \subsetneq Y \wedge \forall x \forall y \big(y < x \wedge X(x) \wedge Y(y) \rightarrow X(y)\big); \\ \mathrm{Suffix}(X,Y) &:= X \subsetneq Y \wedge \forall x \forall y \big(y > x \wedge X(x) \wedge Y(y) \rightarrow X(y)\big), \end{aligned}$$

where $X \subsetneq Y$ means that X is a proper subset of Y. Thus under prefix and suffix relations we always mean proper prefix and suffix.

Two (or more) subtrees of a tree are said to be *sibling subtrees* if their roots have the same parent.

Lemma 4.21 ([Ném06]) *Suppose that P is a (finite or infinite) constructible biposet and X is a subset of P, then*

(i) X is a clan of P if and only if there are consecutive sibling subtrees in P^{tr} such that X is exactly the union of the sets of leaves of these subtrees;

(ii) X is a prime clan of P if and only if X is the set of leaves of a single subtree of P^{tr}.

Proof. We will begin with a proof of case (i). The necessity of the condition is based on the following observation. Suppose that x and y are vertices of P. As we mentioned earlier, we can regard them as two leaves in the tree representation P^{tr}. The (horizontal or vertical) type of the order relation between x and y is solely determined by the label of their lowest common ancestor node. For this reason, let u denote the lowest common ancestor of x and y. If the label of u in P^{tr} is $\bullet$, then $x <_h y$ or $y <_h x$; if the label is $\circ$, then x and y are ordered vertically. We can also easily decide whether x is less or greater than y. Consider u_x and u_y, the children of u that are ancestors of x and y, respectively. Now x is less than y if and only if u_x is less than u_y according to the order of the children nodes at u. It follows that if a subset X of P satisfies the condition

of (i), then it also fulfills the requirements of being a clan. Indeed, the order relation between a vertex x from X and a vertex y outside X is independent of the choice of x from X.

For the converse direction, let us suppose, on the contrary, that X is a clan, but the condition does not hold. First let v denote the lowest common ancestor node of the vertices of X. Now consider those children of v that have leaves in X, and then take the subtrees generated by them. The condition can be violated in two ways. Either these subtrees are not consecutive or there is a subtree that has leaves both from X and $P \setminus X$. In the first case, it is straightforward to show that X is not a clan. In the second case, consider a child u of v that has a leaf x in X and has a leaf z in $P \setminus X$ as well. We can even assume that x and z are descendants of different children of u, so their lowest common ancestor is u. Moreover, there must also be a vertex y in X whose lowest common ancestor with x is v, otherwise the lowest common ancestor of the set X could not be v. But x and z are related by an order of type determined by the label of u, while y and z are related by an order of type determined by the label of v. As u is a child of v, their labels in P^{tr} are different. Consequently, the types of the order relations between x and z and between y and z are also different. This contradicts our assumption that X is a clan.

Now case (ii) easily follows from case (i), since if X consists of the leaves of several (but not all) subtrees of a given node, then overlapping clans can be constructed by (i), showing that X is not prime. □

The following lemma is important for a later proof. It relates the biposet, tree and term representations of finite biwords with the use of parentheses of a PA in a run. Recall that $E'(\Sigma) = \Sigma \cup \{ \bullet, \circ, \langle, \rangle \}$.

Lemma 4.22 ([Ném06]) *For any $P \in \mathrm{SPB}(\Sigma)$, $X \subseteq P$, parenthesizing automaton $\mathcal{A}$, and $\mathbf{r} \in \mathrm{Runs}(\mathcal{A})$ with $P = \mathrm{Label}(\mathbf{r})^{bp}$, the following statements are equivalent:*

(i) X is a proper prime clan of P.

(ii) X is the set of leaves of a proper subtree of P^{tr}.

(iii) P^{tm} can be written as $P^{tm} = u\langle X^{tm}\rangle v$, where $u, v \in E'(\Sigma)^*$, and the subword X^{tm} above corresponds to those vertices of P that are in X. [3]

(iv) $\mathbf{r}$ is of the form $\mathbf{r} = \mathbf{r_1} t_1 \mathbf{r_x} t_2 \mathbf{r_2}$, where $\mathbf{r_1 r_2} \neq \varepsilon$, t_1 and t_2 form a matching parenthesizing transition pair in $\mathbf{r}$, and $\mathbf{r_x}$ denotes the direct subrun of $\mathbf{r}$ on the vertices of X.

[3] Note that the subword X^{tm} can also appear at other places in the word P^{tm}.

Proof. The equivalence of (i) and (ii) follows from Lemma 4.21. The equivalence of (ii) and (iii) is obvious as it expresses a typical correspondence between the term and the tree representations. Finally, the equivalence of (iii) and (iv) is simply a consequence of Lemma 3.12. □

Now we are ready to begin the proof of the main theorem here.

Proof of Theorem 4.18. As we saw in Section 3.11, for binoid languages the equivalence of MSO-definability and recognizability (and hence regularity) directly follows from an analogous equivalence result on text languages shown by Hoogeboom and ten Pas [HtP97]. Even though, here we will outline an alternative proof of this fact, since it will serve as the basis for the proof of the infinite case. Our argument does not rely on the equivalence of recognizability and MSO-definability of finite binary trees, but shows how the operations of parenthesizing automata can be described by logical formulas. Thus we start with the finite case, i.e. with sp-biposets and finite biwords, and explain the necessary changes for the infinite case later on.

First, let $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ be a parenthesizing automaton accepting a binoid language L. Our aim is to construct an MSO-formula φ for which $L_\varphi = L$.

The proof of Lemma 3.23 implies that we may assume that $\mathcal{A}$ accepts biwords via direct and singleton runs only. Therefore, it is sufficient to construct a formula $\varphi_{i,f}$ which expresses the fact that $\mathcal{A}$ has a direct run from an initial state i to a final state f.

We use second order variables to store information about the states of the runs. To be more precise, two types of monadic second order variables will be used. First, let X_s be a variable for each state s in S, and let $Z_{\langle j\rangle_j}$ denote a variable for each pair of parentheses in Ω. Formally,

$$\mathrm{SOVar}_{\mathcal{A}} := \{\, X_s \mid s \in S \,\} \cup \{\, Z_{\langle j\rangle_j} \mid \langle j, \rangle_j \in \Omega \,\}.$$

The general form of $\varphi_{i,f}$ is the following

$$\varphi_{i,f} := \exists X_{s_1} \exists X_{s_2} \dots \exists X_{s_m} \exists Z_{\langle 1\rangle_1} \exists Z_{\langle 2\rangle_2} \dots \exists Z_{\langle n\rangle_n} \psi_{i,f},$$

where $\psi_{i,f}$ expresses the fact that the values of our variables actually encode a direct run of $\mathcal{A}$ from i to f.

We need to check three conditions. First, the run must start from i. Second, it must end in f. Third, we need to be sure that $\mathcal{A}$ has correct transitions everywhere between the states indicated by the variables. We will handle the labeling and the parenthesizing transitions of the run separately.

For labeling transitions, the usual technique (cf. [Str94]) can be applied, that is, the intended meaning of $X_s(x)$, i.e. $x \in X_s$, is that $\mathcal{A}$ reads position x in state s. The storage of parenthesizing transitions is more involved. Fortunately, by Lemma 4.22 the use of parentheses is always around proper prime clans. But we also need to arrange a unique position in the biposet for each matching parenthesizing transition pair used during the run.

For this purpose, the following rule can be applied. If a proper prime clan is a prefix of another proper prime clan, then let the *designated position* be its last position; otherwise let the designated position be its first position. Thus the statement that z is the designated position of a proper prime clan X can be expressed as:

$$\begin{aligned} \mathrm{Dp}(z,X) \;:=\; & \big[\neg\exists Y\big(\,\mathrm{PPC}(Y) \wedge \mathrm{Prefix}(X,Y)\,\big) \wedge \mathrm{First}(z,X)\big] \\ & \vee\big[\,\exists Y\big(\,\mathrm{PPC}(Y) \wedge \mathrm{Prefix}(X,Y)\,\big) \wedge \mathrm{Last}(z,X)\,\big], \end{aligned}$$

where $\mathrm{PPC}(Y)$ states that Y is a proper prime clan, and $\mathrm{First}(z,X)$ ($\mathrm{Last}(z,X)$) is true if and only is z is the first (last, resp.) position of the clan X.

Now it can be verified that the prime property implies that the designated positions of any two proper prime clans never coincide.

Lemma 4.23 ([Ném06]) *Different proper prime clans have different designated positions.*

Proof. We will start by assuming this is not so, and show that it leads to a contradiction. So first assume that X and Y are different proper prime clans, but z is their common designated position. If z is the first position of both X and Y, then either X is a prefix of Y, or Y is a prefix of X, which contradicts the definition of a designated position. If z is the first position of one clan and the last position of the other clan, then X and Y overlap, which again leads to a contradiction. Finally, assume that z is the last position of both X and Y, and $X \subseteq Y$. By definition, there is a proper prime clan X' such that X is a prefix of X'. Therefore, in this case, Y and X' overlap – which is again a contradiction. □

Proof of Theorem 4.18, continued. In the following, we shall assume that no automaton has two opening or closing parenthesizing transitions with the same label. This can easily be achieved by replacing the multiple occurrences of the same parenthesizing transition pair with new transitions using different symbols.

If x is a position, then the intended meaning of $Z_{\langle_j,\rangle_j}(x)$ is that $\mathcal{A}$ uses parentheses $\langle_j,\rangle_j$ (more precisely, the unique pair of transitions labeled $\langle_j$ and $\rangle_j$) before and after processing the proper prime clan whose designated position is x.

As usual, we require that every position belongs to exactly one X_s:

$$\psi_1 \;:=\; \forall x\big[\bigvee_{s\in S} X_s(x) \;\wedge \bigwedge_{\substack{q_1,q_2\in S,\\ q_1\neq q_2}} \big(\neg X_{q_1}(x)\vee\neg X_{q_2}(x)\big)\big].$$

Moreover, the designated positions of proper prime clans must also belong to a unique set $Z_{\langle i\rangle_i}$:

$$\psi_2 := \forall x\Big[\exists X\big(\mathrm{PPC}(X)\wedge \mathrm{Dp}(x,X)\big) \rightarrow \bigvee_{\langle j,\rangle_j\in\Omega} Z_{\langle j\rangle_j}(x) \;\wedge \bigwedge_{\substack{\langle j,\rangle_j\in\Omega,\\ \langle k,\rangle_k\in\Omega,\\ j\neq k}} \big(\neg Z_{\langle j\rangle_j}(x)\vee\neg Z_{\langle k\rangle_k}(x)\big)\Big].$$

We know that for all positions x, the state of $\mathcal{A}$ before processing this position is indicated by a unique state p such that $x \in X_p$. And q, *the state after reading position* x, can be computed as follows. First we observe whether P has a proper prime clan that ends at x. If so, then we determine the smallest such clan, and q is the starting state of the closing parenthesizing transition of that clan. Of course, this state can be determined by observing the designated position of the clan. If there is no proper prime clan that ends at x, then q is indicated at the position $x+1$ or at the greatest prime clan that starts at $x+1$. In addition, if x is the last position, then q must be the final state of the run. Finally, we can check whether $\mathcal{A}$ in fact has a labeling transition with the label of x between p and q. We should do this for every position x. The precise algorithm of this computation and the way of converting it to an MSO-formula are given in the Appendix.

We can also check the correctness of the parenthesizing transitions by a similar procedure. For all proper prime clans, we compute four states of the encoded run: the states before and after the opening, and before and after the closing parenthesizing transitions around the clan. In the pseudocode presented in the Appendix, these states are denoted by *ob*, *oe*, *cb* and *ce*. Then, it is straightforward to check whether $\mathcal{A}$ has a parenthesizing transition pair between the computed states, and whether the labels of these transitions are indicated at the designated position of the clan. It is also a nontrivial computation, since we must take into consideration various inclusion relations of the clans. For more details, see the Appendix once again.

Finally, note that our verification algorithm can be transformed into an MSO-formula ψ_3. Hence we can write $\psi_{i,f}$ as $\psi_{i,f} := \psi_1 \wedge \psi_2 \wedge \psi_3$. This completes the proof for finite constructible biposets.

We will now turn to a brief discussion of the infinite case. Here we will only describe the slight modifications needed compared to the finite case. First, the adaptation of Lemma 4.22 to infinite constructible biposets can be done in the following way. Note that we must distinguish between finite and infinite clans, but this distinction is in parallel with the use of the parenthesizing and separating parentheses.

Lemma 4.24 ([Ném06]) *For any $P \in \mathrm{ISPB}(\Sigma)$, $X \subseteq P$, parenthesizing Büchi-automaton $\mathcal{A}$, infinite run $\mathbf{r} \in Runs(\mathcal{A})$ with $\mathrm{Label}(\mathbf{r})^{bp} = P$, the following statements are equivalent:*

(i) X is a finite proper prime clan of P.

(ii) X is the set of leaves of a finite proper subtree of P^{tr}.

(iii) P^{tm} can be written as $P^{tm} = u\langle X^{tm}\rangle v$, where $u, v \in E'(\Sigma)^*$, and the subword X^{tm} above corresponds to those vertices of P that are in X.

(iv) $\mathbf{r}$ is of the form $\mathbf{r} = \mathbf{r_1} t_1 \mathbf{r_x} t_2 \mathbf{r_2}$, where t_1 and t_2 is a matching parenthesizing transition pair, and $\mathbf{r_x}$ denotes the direct subrun of $\mathbf{r}$ on the vertices of X.

Moreover, the following statements are also equivalent.

(i') X is an infinite proper prime clan of P.

(ii') X is the set of leaves of an infinite proper subtree of P^{tr}.

(iii') P^{tm} can be written as $P^{tm} = u[X^{tm}$, where $u \in E'(\Sigma)^*$, and the subword X^{tm} above corresponds to those vertices of P that are in X.

(iv') $\mathbf{r}$ is of the form $\mathbf{r} = \mathbf{r_1} t \mathbf{r_x}$, where $\mathbf{r_1} \neq \varepsilon$, t is a separating transition of $\mathcal{A}$, and $\mathbf{r_x}$ denotes the direct subrun of $\mathbf{r}$ on the vertices of X.

Now we return to the final part of the proof of the main theorem here.

Proof of Theorem 4.18, completed. It is trivial that we can express the finiteness of clans, as

$$\mathrm{Finite}(X) \quad := \neg\exists z\, \mathrm{Last}(z, X).$$

Hence we can easily locate the separating transitions and check their correctness as well. Furthermore, we have no trouble in formulating the acceptance condition: a finite state has to appear infinitely often as an outer state of the encoded run. We will leave the reader to verify the correctness of the formulas below.

$$\begin{aligned}
\psi_{\mathrm{acc}} := \bigvee_{f\in F} \forall z\, \exists X \Big[& \mathrm{MaxFiniteClan}(X) \wedge \mathrm{OuterState}_f(X) \\
& \wedge \forall x \big(\mathrm{Last}(x, X) \rightarrow (z < x) \big) \Big]; \\
\mathrm{MaxFiniteClan}(X) := & \mathrm{Finite}(X) \wedge \mathrm{Clan}(X) \\
& \wedge \neg\exists Y \big(\mathrm{Finite}(Y) \wedge \mathrm{Clan}(Y) \wedge X \subsetneq Y \big); \\
\mathrm{OuterState}_f(X) := & \big[\mathrm{Singleton}(X) \wedge \exists x \big(X(x) \wedge X_f(x) \big) \big] \\
& \vee \big[\mathrm{PPC}(X) \wedge \bigvee_{\substack{(f,\langle_k,p)\in\gamma \\ \langle_k\in\Omega, p\in S}} \exists z \big(\mathrm{Dp}(z, X) \wedge Z_{\langle_k\rangle_k}(z) \big) \big].
\end{aligned}$$

Of course, the formulas $\mathrm{Finite}(X)$, $\mathrm{Clan}(X)$ and $\mathrm{Singleton}(X)$ have their expected meanings here. □

Finally, we can summarize the main results of this chapter in a concise way.

Theorem 4.25 ([Ném06]) *Let $L \subseteq \Sigma^\omega(\bullet, \circ)$. Then L is recognizable if and only if L is regular if and only if L is* MSO*-definable. Thus* $\mathsf{Rec}^\omega = \mathsf{Reg}^\omega = \mathsf{MSO}^\omega$.

Chapter 5

Conclusions

Finally, I would like to present three things. First I would like to summarize the main contributions in a series of points. Second I will draw some general conclusions. Then I would like to outline some open problems and possible future directions of research.

5.1 Thesis Contributions

- In the thesis I investigated two-dimensional languages as a generalization of classical word languages. The generalization is based on an algebraic approach namely I considered elements of the free binoids – called biwords – instead of words, which are elements of the free monoids.

- Biwords can be represented in a numbers of ways. Here I investigated sp-biposet, two-dimensional word, term, condensed term, ordered unranked tree, i-term, i-cterm ($i \geq 1$), and alternating text representations.

- I introduced parenthesizing automata whose function is to accept binoid languages and to define the class of regular binoid languages. Then I proved that the expressive power of PA is the same as that of algebraic recognizability and second-order monadic definability, i.e. $\mathsf{Reg} = \mathsf{Rec} = \mathsf{MSO}$. After I elaborated on the operation of a PA from different perspectives. Naturally, given a PA and a biword, we have a nondeterministic linear time algorithm which decides whether the automaton accepts the biword. Furthermore, parenthesizing automata are suitable for exploring combinatorial properties of recognizable binoid languages.

- With the help of the PA concept I also demonstrated that regular binoid languages

are closed under ξ-substitution, which immediately implies some other closure properties – namely closure under the products, iterations and homomorphisms.

- Recall the Reg_i denote those binoid languages that can be accepted by a PA with at most i pairs of parentheses symbols ($i \geq 0$). I found that these languages classes form a strict hierarchy, that is

$$\mathsf{Reg}_0 \subsetneq \mathsf{Reg}_1 \subsetneq \mathsf{Reg}_2 \subsetneq \dots$$

- I investigated various classes of rational binoid languages (HRat, VRat, BRat, GRat), horizontally (HB) and vertically bounded (VB) languages, along with binoid languages of bounded depth (BD). I established the following relationships among them.

$$\mathsf{BRat} = \mathsf{Reg} \cap \mathsf{BD} = \mathsf{Reg}_1 \cap \mathsf{BD},$$
$$\mathsf{HRat} = \mathsf{Reg} \cap \mathsf{VB},\ \mathsf{VRat} = \mathsf{Reg} \cap \mathsf{HB} \text{ and}$$
$$\mathsf{HRat} \cup \mathsf{VRat} \subsetneq \mathsf{BRat} \subsetneq \mathsf{GRat} \subsetneq \mathsf{Reg}.$$

- I extended my investigations to ω-biwords. I presented a free algebra theorem and a graph-theoretic (or order-theoretic) characterization for them.

- I defined parenthesizing Büchi-automata operating on ω-biwords and with the help of them I demonstrated that the equivalence of regularity, recognizability and MSO-definability holds for ω-binoid languages as well, i.e. $\mathsf{Reg}^\omega = \mathsf{Rec}^\omega = \mathsf{MSO}^\omega$.

- All the results can be generalized to higher dimensions, i.e. to free algebras where three or more independent associative operations are present.

5.2 General Conclusions

Some specific conclusions were already drawn in Section 3.12.3. Now we will have a brief discussion about the effectiveness of the characterization, about our new automata model and about a way of generalizing our theory to higher dimensions.

First it should be mentioned that all of our characterizations of the classes Reg, Reg^ω and $\mathsf{Reg} \cap \mathsf{BD}$ are effective in the sense that, from any realization (by an automaton, a recognizing (ω)-binoid, an MSO-formula or a rational expression) of a concrete language, any equivalent other realization can be constructed via an algorithm. For instance, for a PA an equivalent monoid automaton (for the T_i-mode or C_i-mode) can be constructed,

and vice versa. Furthermore, this transition algorithm increases the number of states of the automaton by just a constant factor. Of course, the conversion of a nondeterministic PA to equivalent recognizing binoids requires exponential time, since this is already unavoidable in the word case.

If we look at a PA as a model of computations an disadvantageous feature about it arises. It is that a PA is not a finite state device, at least in the strict sense. Indeed, in Section 3.2.3, where configurations and a transition relation among them were defined, we saw that the second component of a configuration serves as a pushdown stack. It stores the opened (but not yet closed) parentheses. In fact it stores the parentheses together with their indices given by the parenthesizing transition that was used when the parentheses was processed. Hence a PA obviously possesses some 'internal memory' with a pushdown behavior, whose size is not limited by the automaton. Of course, during a concrete computation on a biword w, the size of the stack is equal to the depth (see Definition 3.43) of w.

However it should be noticed that the nonregularity of the Dyck word languages (see Section 2.6.7) implies that, if the acceptance is based on the term or cterm representation of biwords, it cannot have the finite state property. Of course in theory it is possible to build an accepting device on another representation of a biword, e.g. on sp-biposets. But sp-biposets themselves are not linear objects, and if we wish to consider the linear order determined by $<_h \cup <_v$, we get back the same linear order given by the term representation, i.e. the ordering of the letters when reading them from left to right.

Next, let us say a few words about the generalization of our theory to three or more dimensions. Let n be any positive integer greater than two. In Sections 2.2 and 2.3 n-semigroups and n-monoids were introduced as the kind of algebras where n associative operations are defined. In addition, n-monoids have a common identity elements for all their operations. Afterwards, for any alphabet Σ we can consider the free n-monoids over Σ whose elements can be called *n-words*.

The free algebra theorem, i.e. the representation of n-words by constructible n-posets was stated in Theorem 2.11. As for the other representations, it is straightforward to extend the two-dimensional word, term and tree representations of biwords. Of course, for the latter two, n operation symbols like $\circ_1,\ldots,\circ_n$, are needed. But for the generalization of the cterm representation, a single pair of parenthesis is insufficient, since the indices of the operation symbols in an n-word ($n \geq 3$) are not always alternates among the parentheses as in the biword case. But one can still employ n pairs of parentheses to reconstruct the type of the operations, and hence achieve a condensation.

For instance the 3-word $\langle a \circ_3 \langle b \circ_1 c\rangle\rangle \circ_2 d$ can be reduced to $\circ_2 \langle {}_3a\langle {}_1bc\rangle_1\rangle_3 d$.

The generalization of recognizability and MSO definability for n-monoid languages is trivial. As for regularity, we need to change the parenthesizing automata models just sightly. Actually the states of automata need to be divided into n sets $S_1, S_2, \ldots, S_n$, not just two. Moreover, parenthesizing transitions now can take place between any two states from any two distinct sets. In addition, during the operation of the n-dimensional automata we require that if a run goes through a state from the set S_i, then the $\circ_i$ operation should be applied between the labels of the subruns appearing before and after that state.

Although the generalization of texts with n linear orders is of course possible, it seems that we cannot get a generalization of the isomorphism established between alternating text and sp-biposets in Lemma 3.69. Nevertheless the method presented in Section 4.8 can still be used to prove the equivalence between regularity and MSO-definability for n-monoid languages.

Lastly, it is not hard to see that with these changes the generalized versions of Theorem 3.35 and Theorem 3.70 can be proved.

5.3 Open Problems and Further Directions

I cannot deny that the results of this thesis are really just the first steps in the investigations of binoid languages. Not surprisingly several problems remain open. In the following we will briefly outline some of them. This list can also be viewed as a form of self-criticism stating what else should have been done and should be done in the future. Actually, I learnt some of these points from the referee's reports of my publications, for which I am most thankful.

- We managed to generalize the equivalence of regularity, recognizability and MSO-definability from word languages to binoid languages, but we only succeed in defining an equivalent concept of rationality in the bounded depth case. However our investigation showed that $\mathsf{GRat} \subsetneq \mathsf{Reg}$, i.e. the two product operations and the two iterations even with the complementation is insufficient to construct all regular binoid languages from finite binoid languages. On the other hand, if we allowed ξ-iteration, it would generate nonregular binoid languages, as the following classical example shows:

$$\{a \bullet \xi \bullet n\}^{*\xi} = \{a^{\bullet n} \bullet \xi \bullet b^{\bullet n} \mid n \geq 0\}.$$

  Here the ξ-iteration of a binoid language is defined in the same way as for tree

languages (cf. [GS84]), i.e. $L^{*\xi} = \cup_{i\geq 0} L_i^\xi$, where $L_0^\xi = \{\xi\}$, and $L_{i+1}^\xi = L_i^\xi[L/\xi] \cup L_i^\xi$. Now the question is whether the ξ-iteration has a kind of restriction which captures regularity, or whether there is any other operation which (with the operation of BRat or GRat) generates all regular languages from the finite binoid languages. Recall that the classical Kleene theorem describes the behavior of finite automata by the operations of union, concatenation and iteration. Therefore, put another way, we would like to now what the operations on binoid languages are which capture the behavior of parenthesizing automata.

- Another open problem that seems to be difficult to solve is the decidability status of the question whether a given regular language appears at certain level of the hierarchy of the classes Reg_i, $i \geq 0$. It would also be interesting to know whether the levels of this hierarchy can be characterized logically or algebraically.

- In the thesis we did not deal with first-order definable binoid languages. Their decidability and algebraic characterization are open problems as well. Perhaps it is also possible to do this in terms of parenthesizing automata. It should be mentioned here too that there is a natural correspondence between the class of generalized binoid languages (GRat) and the class of star-free word languages. Moreover, in [ÉN04] the notion of aperiodic binoids was introduced. There it was shown that every binoid language in GRat can be recognized in an aperiodic binoid. However as for the converse statement – namely whether every binoid language recognized in an aperiodic binoid is generalized birational – is currently an open question.

- Two fundamental algorithms of classical automata theory are the determinization and minimization algorithm of automata. Can they be extended to parenthesizing automata? It is not hard to define deterministic PA and show that every nondeterministic PA is equivalent to a deterministic one. Indeed, from the proof of $\mathsf{Rec} \subseteq \mathsf{Reg}$ (Theorem 3.35) we get a PA that can be regarded as deterministic. This is not surprising at all, as recognizability is clearly a deterministic notion. However this leads to just an indirect algorithm of determinization: first one needs to transform a nondeterministic PA to an equivalent binoid, and then transform it back to a deterministic PA. Notice that this procedures increase exponentially not just the states of the original automaton (which is, of course, unavoidable) but also the number of parenthesis symbols. Is there any direct algorithm which is better in this aspect?

- During the construction of infinite biword we applied some serious restrictions: we

only allowed to form the (horizontal or vertical) product of two finite biwords, or a finite biword with an infinite one. Also the ω-product operations were applied only to finite biwords. This restriction seem to be necessary for our proofs. However it has been shown that classical automata can be extended from words (i.e. from labeled finite linear orders) not just to ω-words, but also to ordinal words and even to any countable linear orders [BC07]. Is it possible to generalize parenthesizing automata in this way too?

- Finally it would also be good to look for concrete applications of our theory. Since the special features of biwords and their n-dimensional generalizations is that they are naturally equipped with some *nested structures*, it seems obvious to look for applications where some nestedness (of arbitrary depth) is present, e.g. in XML databases and in modeling recursive function calls. Concrete applications of PA theory would demonstrate the usefulness of it, help us to see its connections with the real word, and provide insights into its detailed workings.

Summary

In this thesis we laid the foundations for a two-dimensional theory of automata and languages. For the generalization from the one-dimensional case of words we adopted an algebraic approach, namely we considered languages over free binoids. It is a generalization of monoids where two independent associative operations are defined and they share a common identity element. We managed to generalize the equivalence of regularity, recognizability and MSO-definability from word languages to binoid languages and to ω-binoid languages as well. We also introduced various concepts of rational binoid languages and examined their relationships. All the results can be generalized to higher dimensions, i.e. to free algebras where three or more independent associative operations are present.

The first chapter gave an introduction to the general notion of regularity. We outlined preliminary studies relevant to our investigations, stated the main aims of the thesis and cited related literature. In addition, we enumerated our key results and we briefly outlined the structure of the thesis.

In Chapter 2 we dealt with a two-dimensional algebraic generalization of words called biwords. In Section 2.1 we recalled some basic notions frequently used in the theory of formal languages and algebra, and we introduced some notations. In the next two sections we investigated those free algebras whose elements we worked with – namely free bisemigroups and binoids. Before discussing free binoids and their elements, we considered the more general framework of (m, n)-semigroups and (m, n)-structures introduced by Ésik. This general setting allows one to consider any finite number of operations that are associative, or associative and also commutative. First in Section 2.4 we presented the proof of a theorem in [Ési00] which states that the elements of the free (m, n)-semigroups can be described by so-called reducible (m, n)-structures. As a special case of this result we also obtained a description of the elements of the free n-semigroups (resp. bisemigroups) by constructible n-posets (resp. by series-parallel biposets). Section 2.6 was essential for the subsequent chapters, since the representa-

tions of biwords – the elements of the free binoids – were studied there. Namely, we introduced sp-biposet, term, condensed term, two-dimensional word, tree, i-term and i-cterm representations of biwords. We gave a characterization of those words that are i-terms or i-cterms (Lemma 2.19 and Lemma 2.20). Then we discussed the properties and the suitability of the various representations. In Section 2.7, based on [Ésik00], a graph-theoretic (or order-theoretic) characterization of the elements of the free (m, n)-semigroups, n-semigroups and bisemigroups was given (Theorem 2.25, Corollaries 2.26 and 2.27).

In Chapter 3 we studied languages of finite biwords and parenthesizing automata which operate on them. In Section 3.1 the novel concept of parenthesizing automata was introduced (Definition 3.1), which was then illustrated by a simple example. In Section 3.2 the operation of parenthesizing automata was discussed. We considered three approaches, and afterward we proved that they are in fact equivalent. Thus any of them is suitable for defining the concrete binoid language accepted by an automaton, and hence the class of regular binoid languages. Section 3.3 contains some examples for parenthesizing automata and regular binoid languages. These aided our understanding and appeared in later proofs as well. In Section 3.4 we demonstrated the necessity of forbidding double parenthesization in automata and the technical difficulties it causes. In Section 3.5 the notion of a substitution product (Definition 3.20) was introduced to overcome these difficulties. Next, in Section 3.6 we studied normal forms of parenthesizing automata, and proved that for each parenthesizing automaton $\mathcal{A}$ there exists an equivalent automaton $\mathcal{A}_h$ (resp. $\mathcal{A}_v$) which has a single horizontal (resp. vertical) pair of initial and final state. In Section 3.7 we showed that the class of regular binoid languages is effectively closed under the operation of ξ-substitution (Theorem 3.25). This result implies some other closure properties, namely the closure under the horizontal and vertical products and horizontal and vertical iterations as well as under homomorphisms. In Section 3.8 we gave a more refined classification of regular binoid languages by showing that the classes Reg_m of languages accepted by a parenthesizing automaton with at most m pairs of parentheses form a strict hierarchy (Theorem 3.32). In fact, this hierarchy is proper for all alphabets (Theorem 3.33). Hence the number of parentheses needed to accept a given binoid language provides a complexity measure on the class of regular binoid languages. Sections 3.9 and 3.11 served to demonstrate that the notion of parenthesizing automata was well chosen. There it was shown that the class of regular binoid languages coincides both with the class of algebraically recognizable languages (Theorem 3.35) and with the class of languages definable in second-order monadic logic

(Theorem 3.70). For the logical characterization we applied a similar result of Hoogeboom and ten Pas on text languages (Theorem 3.65). For this we introduced the notion of texts in Section 3.11.2, and in Section 3.11.3 we proved an isomorphism between sp-biposets and alternating texts. In Section 3.10 we considered several *rational* classes of binoid languages and studied the inclusion relations among them, and also their relations to the class of regular languages and to the classes of horizontally/vertically bounded and bounded depth binoid languages. The chapter closed with Section 3.12, where we established a detailed comparison between the concept of regularity of ours and that of Hashiguchi et al. Their notion of regularity turned out to be less general than ours. After we managed to extend their monoid approach to the broader class of our regular binoid languages. This means that with appropriate definitions ordinary finite automata are also capable of capturing the same concept of regularity. This provided a fourth equivalent characterization of our class Reg of regular binoid languages.

In Chapter 4 languages of infinite biwords were investigated. First in Section 4.1 we defined ω-biwords and algebras needed for their algebraic characterizations, namely ω-semigroups in the pattern of classical ω-words and ω-semigroups of Perrin and Pin [PP04]. The free algebra theorem (Theorem 4.2) and the graph-theoretic characterization of the elements of the free ω-bisemigroups (Theorem 4.3) are found there as well. Section 4.2 then studied the tree and term representations of ω-biwords. It was followed by an extension of recognizability, MSO-definability and regularity to ω-binoid languages in Sections 4.3, 4.4 and 4.5, respectively. To extend regularity we needed to define the concept and the operation of parenthesizing Büchi-automata. The rest of the chapter was devoted to the proof of the equivalence of the above three concepts. In more detail, in Section 4.6 it was shown that regularity implies recognizability. Similarly in Section 4.7 we proved that regularity follows from recognizability. Then in Section 4.8 the equivalence of regularity and MSO-definability was demonstrated. The technical details of a proof in Section 4.8, namely an algorithm which creates an equivalent logical formula from a parenthesizing Büchi-automaton, can be found in the Appendix.

Finally, in Chapter 5 we gave a brief overview of and discussion about the main contributions of the thesis and we also stated some open problems and suggested possible future directions of research.

Összefoglalás
(Summary in Hungarian)

Az értekezésben leraktuk a kétdimenziós szavak és automaták egy lehetséges elméletének alapjait. A szavak egydimenziós esetének általánosításához egy algebrai megközelítés által jutottunk, nevezetesen szabad binoidok feletti nyelveket tekintettünk. A szabad binoidok a szabad monoidok azon általánosításai, melyekben egy helyett két független asszociatív művelet van értelmezve, továbbá ezeknek a műveleteknek közös egységelemük van. Sikerült általánosítanunk a regularitás, a felismerhetőség és az MSO-definiálhatóság ekvivalenciáját szavakról binoid nyelvekre és ω-binoid nyelvekre is. Különböző racionális binoid nyelvosztályokat is definiáltunk és elemeztünk. Eredményeink általánosíthatók tetszőleges magasabb dimenziószámra, azaz olyan szabadalgebrák részhalmazaira, mely algebrákban három vagy több független asszociatív művelet van.

Az első fejezet betekintést nyújtott a regularitás általános fogalmába. Majd vázoltuk kutatásunk előzményeit, legfontosabb célkitűzéseit, valamint a kapcsolódó szakirodalmat. Ezen túl felsoroltuk legfontosabb eredményeinket, majd röviden ismertettük az értekezés szerkezetét.

A második fejezetben a szavak kétdimenziós algebrai általánosításával, a biszavakkal foglakoztunk. A 2.1. alfejezetben néhány, a formális nyelvek elméletében és az algebrában általánosan használt alapfogalom definícióját idéztük fel, illetve bevezettünk néhány jelölést. A következő két alfejezet azokat az algebrai struktúrákat ismertette, melyek szabadalgebráival dolgoztunk, nevezetesen a bifélcsoportokat és a bimonoidokat. De mielőtt a szabad binoidok elemeinek tárgyalására rátértünk volna, a 2.4. fejezetben az Ésik által bevezetett (m, n)-félcsoportok és (m, n)-struktúrák általánosabb fogalmait vettük át. (Az (m, n)-félcsoportok általános kerete lehetővé teszi, hogy tetszőleges véges számú asszociatív, illetve asszociatív és egyszersmind kommutatív műveletet tartsunk szem előtt.) A 2.5. alfejezetben először annak a bizonyítását mutattuk be ([Ési00] alapján), hogy az ún. redúcibilis (m, n)-struktúrák alkotják a szabad (m, n)-félcsoportokat. Majd ennek speciális eseteként kaptuk a szabad n-félcsoportok (illetve bifélcsoportok) eleme-

inek leírását ún. konstruálható n-posetekkel (illetve sp-biposetekkel). A 2.6. alfejezet elengedhetetlenül fontos a későbbi fejezetekhez, mivel itt tárgyaltuk a binoid nyelvek elemeinek, a biszavaknak, a különböző reprezentációit. Nevezetesen ismertettük a biszavak soros-párhuzamos biposet (sp-biposet), term, kétdimenziós szó (two-dimensional word), tömörített term (condensed term), fa (tree), i-term és i-cterm reprezentációit. Ezt követően jellemzést adtunk azokra a szavakra, melyek i-termek illetve i-ctermek (2.19. és 2.20. Lemma). Végül kitértünk a különböző reprezentációk sajátosságaira és alkalmazhatóságára. A 2.7. alfejezetben az sp-posetek és kográfok (cographs) tiltott részgráfokkal (nevezetesen „N"-mentes, illetve P_4-mentes gráfokként) történő jelemzéseinek [Gra81, VTL82, CLB81] általánosításait ismertettük. Először a reducibilis (m, n)-struktúrákét (2.25. Tétel, [Ési00] alapján), majd ennek következményeként a konstruálható n-posetekét (2.26. Következmény) és végül a soros-párhuzamos biposetekét (2.27. Következmény).

A 3. fejezet véges biszavakból álló nyelvekkel és az őket feldolgozó zárójelező automatákkal foglalkozott. A 3.1. alfejezetben bevezettük a zárójelező automata fogalmát (3.1. Definíció), melyet egy egyszerű példával illusztráltunk. A 3.2. alfejezetben a zárójelező automaták működését adtuk meg. Három különböző megközelítést tekintettünk, majd igazoltuk, hogy ezek ugyanahhoz az elfogadás fogalomhoz vezetnek. Ezért bármelyikük alkalmas arra, hogy segítségével egy adott automata által elfogadott nyelvet, valamint a reguláris binoid nyelvek osztályát definiáljuk. A 3.3. alfejezetben néhány példa következett zárójelező automatákra és reguláris binoid nyelvekre. Ezek – azon túl, hogy a fogalom megértését segítették – néhol későbbi bizonyításokban is szerepeltek. A 3.4. alfejezetben az ún. dupla zárójelezés tiltásának szükségességét mutattuk meg, és az ebből adódó technikai nehézségeket vázoltuk. A 3.5. alfejezetben pedig ezen nehézségek leküzdésére szolgált a helyettesítő szorzat (substitution product, 3.20. Definíció) bevezetése. A 3.6. alfejezetben a zárójelező automaták normálformái következtek. Nevezetesen megmutattuk, hogy minden zárójelező $\mathcal{A}$ automatához megadható olyan, vele ekvivalens $\mathcal{A}_h$ (illetve $\mathcal{A}_v$) zárójelező automata, mely egyetlen horizontális (illetve vertikális) kezdő és végállapot párral rendelkezik. A 3.7. alfejezetben bebizonyítottuk, hogy a reguláris binoid nyelvek osztálya effektíven zárt a ξ-helyettesítés műveletére (3.25. Tétel). Ebből az eredményből közvetlenül adódnak további zártsági tulajdonságok. Nevezetesen a reguláris binoid nyelvek zártak a horizontális és vertikális szorzás, valamint a horizontális és vertikális iteráció műveletére, továbbá a homomorfizmusokra nézve. A 3.8. alfejezetben a reguláris binoid nyelvek egy finomabb osztályozását adtuk azáltal, hogy bebizonyítottuk, hogy a legfeljebb $m \geq 0$ zárójel szimbólummal rendelkező zárójelező automatákkal elfogadható binoid nyelvek Reg_m osztályai valódi hierarchiát

alkotnak (3.32. Tétel). Mi több igazoltuk, hogy ez a hierarchia valódi minden rögzített ábécé esetén is (3.33. Tétel). Így a felismeréshez szükséges zárójelek száma a binoid nyelvek leírásának egyfajta bonyolultsági mértékét adja. A 3.9. és a 3.11. alfejezet a bevezetett automata fogalom alkalmasságának igazolására szolgált. Bennük megmutattuk, hogy a reguláris binoid nyelvek osztálya egybeesik mind az algebrailag felismerhető (3.35. Tétel) mind a monadikus másodrendben definiálható (3.70. Tétel) nyelvek osztályával. A logikai jellemzéshez felhasználtuk Hoogeboom és ten Pas text nyelvekre vonatkozó hasonló eredményét (3.70. Tétel). Az ehhez szükséges text nyelvek fogalma a 3.11.2. alfejezetben, az sp-biposetek és text nyelvek kapcsolatát pedig a 3.11.3. alfejezetben ismertettük. A 3.10. alfejezetben különböző racionális kifejezésekkel megadható binoid nyelveket és ezek tartalmazási relációit vizsgáltuk, valamint ezen osztályok viszonyát a horizontálisan/vertikálisan korlátos és a korlátos mélységű nyelvekhez. A fejezetet a 3.12. alfejezet zárja, melyben az általunk definiált zárójelező automatákat vetettük össze a Hashiguchi et al. által bevezetett véges automatákkal történő felismeréssel. Az ő regularitás fogalmuk kevésbé általánosnak bizonyult, mint a miénk. Ezután sikerült az ő, hagyományos automatákon alapuló módszerüket kiterjeszteni a mi reguláris nyelveink bővebb osztályára. Ez a megközelítés adja a reguláris binoid nyelvek osztályának, Reg-nek a negyedik ekvivalens jellemzését.

A negyedik fejezetben végtelen biszavakból álló nyelvekkel foglalkoztunk. Először az ω-biszavakat és a jellemzésükhöz szükséges algebrákat, az ω-bifélcsoportokat definiáltuk a klasszikus eset, vagyis a Perrin és Pin [PP04] munkájában használt ω-szavak és ω-félcsoportok mintájára (4.1. alfejezet). A szabadalgebra tétel (4.2. Tétel) és a szabad ω-bifélcsoportok elemeinek gráfelméleti jellemzése (4.3. Tétel) szintén itt találhatók. A 4.2. alfejezet tárgyalta az ω-biszavak fa és term reprezentációit. Ezután a felismerhetőség (recognizablility), a monadikus másodrendű logikai definiálhatóság (MSO-definiability) és a regularitás fogalmának kiterjesztése következett ω-binoid nyelvekre, rendre a 4.3, a 4.4. és 4.5. alfejezetben. A regularitás kiterjesztéséhez definiálnunk kellett a zárójelező Büchi-automaták fogalmát és működésüket. A fejezet további részeiben a fenti három fogalom ekvivalenciáját bizonyítottuk. A 4.6. alfejezetben azt igazoltuk, hogy a regularitásból következik a felismerhetőség. A 4.7. fejezet arról szólt, hogy a felismerhetőség maga után vonja a regularitást. A 4.8. alfejezetben pedig a regularitás és a monadikus másodrendű logikai definiálhatóság ekvivalenciáját láttuk be. A 4.8. fejezetben szereplő egyik bizonyítás technikai részletei – nevezetesen a zárójelező Büchi-automatából ekvivalens logikai formulát készítő algoritmus – a függelékbe kerültek.

Végezetül az 5. fejezetben egy rövid áttekintést adtunk az értekezés eredményeiről, a megoldatlan problémákról és a további kutatási irányokról.

Bibliography

[AM04] R. Alur, P. Madhusudan, Visibly pushdown languages. In: proc *STOC 2004*, Chicago, IL, USA, June 13-16, 2004, 202–211.

[AM06] R. Alur, P. Madhusudan, Adding nesting structure to words. In: proc *DLT'06*, LNCS **4036**, 1-13.

[BÉ98] S. L. Bloom, Z. Ésik. Shuffle binoids. *Theoret. Inform. Appl.* **32** (1998), 175–198.

[BLS99] A. Brandstädt, V.B. Le, J.P. Spinrad, *Graph classes: a survey*. SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, 1999.

[BC07] V. Bruyère, O. Carton, Automata on linear orderings. *J. Comput. System Sci.* **73** (2007), 1–24.

[BS81] S. Burris, H.P. Sankappanavar, *A course in universal algebra*. Graduate Texts in Mathematics **78**, Springer-Verlag, New York-Berlin, 1981.

[Büc60] J. R. Büchi, Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.* **6** (1960), 66–92.

[CLB81] D. G. Corneil, H. Lerchs, L. S. Burlinham, Complement reducible graphs. *Discr. Appl. Math.* **3** (1981), 163–174.

[Cou91] B. Courcelle, The monadic second-order logic on graphs. V. On closing the gap between definability and recognizability. *Theoret. Comput. Sci.* **80** (1991), 153–202.

[Cou96] B. Courcelle, Basic notions of universal algebra for language theory and graph grammars. *Theoret. Comput. Sci.* **163** (1996), 1–54.

[Cou03] B. Courcelle, The monadic second-order logic of graphs. XIV. Uniformly sparse graphs and edge set quantifications. *Theoret. Comput. Sci.* **299** (2003), 1–36.

[CW05] B. Courcelle, P. Weil, The recognizability of sets of graphs is a robust property. *Theoret. Comput. Sci.* **342** (2005), 173-228.

[DR95] V. Diekert, G. Rozenberg (eds.), *The book of traces.* World Scientific Publishing Co., Inc., River Edge, NJ, 1995.

[Dol05] I. Dolinka, A note on identities of two-dimensional languages. *Disc. Appl. Math.* **146** (2005), 43–50.

[Dol07] I. Dolinka, Axiomatizing the identities of two-dimensional languages. *Theoret. Comput. Sci.* **372** (2007), 1–14.

[Don70] J. Doner, Tree acceptors and some of their applications. *J. Comput. System Sci.* **4** (1970), 406–451.

[ER90] A. Ehrenfeucht, G. Rozenberg, Theory of 2-structures, Part I: Clans, basic subclasses, and morphisms. Part II: Representation through labeled tree families. *Theoret. Comput. Sci.* **70** (1990), 277–303, 305–342.

[ER92] A. Ehrenfeucht, G. Rozenberg, Angular 2-structures. *Theoret. Comput. Sci.* **92** (1992), 227–248.

[ER93] A. Ehrenfeucht, G. Rozenberg, T-structures, T-functions and texts. *Theoret. Comput. Sci.* **116** (1993), 227–290.

[EPR93] A. Ehrenfeucht, P. ten Pas, G. Rozenberg, Combinatorial properties of texts. *Theor. Inf. Appl.* **27** (1993), 433–464.

[Eil76] S. Eilenberg, *Automata, Languages, and Machines.* Vol. B, Academic Press, New York, 1976.

[Elg61] C. C. Elgot, Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.* **98** (1961), 21–51.

[EHPR96] J. Engelfriet, T. Harju, A. Proskurowski, G. Rozenberg, Characterization and complexity of uniformly nonprimitive labeled 2-structures. *Theoret. Comput. Sci.* **154** (1996), 247–282.

[Ésik00] Z. Ésik, Free algebras for generalized automata and language theory. *RIMS Surikaisekikenkyusho Kokyuroku*, Kyoto University, Kyoto, No. **1166** (2000), 52–58.

[ÉN02] Z. Ésik, Z. L. Németh, Automata on series-parallel biposets. In: proc. *DLT'01*, LNCS **2295**, Springer-Verlag, 2002, 217–227.

[ÉN04] Z. Ésik, Z. L. Németh, Higher dimensional automata. *J. of Autom. Lang. Comb.* **9** (2004), 3–29.

[ÉN05] Z. Ésik, Z. L. Németh, Algebraic and graph-theoretic properties of infinite n-posets. *Theoret. Informatics Appl.* **39** (2005), 305–322.

[ÉO99] Z. Ésik, S. Okawa, Series and parallel operations on pomsets. In: proc. *FST & TCS'99*, LNCS **1738**, Springer-Verlag, 1999, 316–328.

[GS84] F. Gécseg, M. Steinby, *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[GS97] F. Gécseg, M. Steinby, Tree languages. In: G. Rozenberg, A. Salomaa (eds.), *Handbook of formal languages*, Vol. 3, Springer-Verlag, Berlin, 1997, 1–69.

[GR96] D. Giammarresi, A. Restivo, Two-dimensional finite state recognizability. *Fund. Inform.* **25** (1996), 399–422.

[GR97] D. Giammarresi, A. Restivo, Two-dimensional languages. In: G. Rozenberg, A. Salomaa (eds.), *Handbook of formal languages*, Vol. 3, Springer-Verlag, Berlin, 1997, 215–267.

[GRST94] D. Giammarresi, A. Restivo, S. Seibert, W. Thomas, Monadic second order logic over pictures and recognizability by tiling systems. In: proc. *STACS'94*, LNCS **775**, Springer-Verlag, 1994, 365–375.

[Gis88] J. L. Gischer, The equational theory of pomsets. *Theoret. Comput. Sci.* **61** (1988), 199–224.

[Gra81] J. Grabowski, On partial languages. *Fund. Inform.* **4** (1981), 427–498.

[Har94] F. Harary, Kuratowski's theorem. In: *Graph Theory*, Addison-Wesley, Reading, Mass., 1994, 108–113.

[Har78] M. A. Harrison, *Introduction to formal language theory.* Addison-Wesley Publishing Co., Reading, Mass., 1978.

[HHJ03] K. Hashiguchi, K. Hashimoto, S. Jimbo. Modified RSA cryptosystems over bicodes. *Advances in algebra,* World. Sci. Publ., NJ, 2003, 377–389.

[HIJ00] K. Hashiguchi, S. Ichihara, S. Jimbo, Formal languages over free binoids. *J. Autom. Lang. Comb.* **5** (2000), 219–234.

[HKJ02] K. Hashiguchi, T. Kunai, S. Jimbo, Finite codes over free binoids. *J. Autom. Lang. Comb.* **7** (2002), 505–518.

[HM99] K. Hashiguchi, T. Mizoguchi, Introduction to bicodes. In: *Algebraic Engineering,* World Sci. Publ., NJ, 1999, 219–229.

[HSJ04] K. Hashiguchi, Y. Sakakibara, S. Jimbo, Equivalence of regular binoid expressions and regular expressions denoting binoid languages over free binoids. *Theoret. Comput. Sci.* **312** (2004), 251–266.

[HWJ03] K. Hashiguchi, Y. Wada, S. Jimbo, Regular binoid expressions and regular binoid languages. *Theoret. Comput. Sci.* **304** (2003), 291–313.

[HtP96] H. J. Hoogeboom, P. ten Pas, Text languages in an algebraic framework. *Fund. Inform.* **25** (1996), 353–380.

[HtP97] H. J. Hoogeboom, P. ten Pas, Monadic second-order definable text languages. *Theory Comput. Syst.* **30** (1997), 335–354.

[ISGCI] Information System on Graph Class Inclusions v2.0, Graphclass: Cographs. `http://wwwteo.informatik.uni-rostock.de/isgci/classes/gc_151.html` (accessed January 16, 2007).

[Kus01] D. Kuske, Infinite series-parallel posets: logic and languages. In: proc. *ICALP 2000*, LNCS **1853**, Springer-Verlag, 2001, 648–662.

[Kus03a] D. Kuske, Towards a language theory for infinite N-free pomsets. *Theoret. Comput. Sci.* **299** (2003), 347–386.

[Kus03b] D. Kuske, Regular sets of infinite message sequence charts. *Inform. and Comput.* **187** (2003), 80–109.

[KM04] D. Kuske, I. Meinecke, Branching automata with costs—a way of reflecting parallelism in costs. *Theoret. Comput. Sci.* **328** (2004), 53–75.

[Lap98] D. Lapoire, Recognizability equals monadic second-order definability for sets of graphs of bounded tree-width. In: proc. *STACS'98*, LNCS **1373**, Springer-Verlag, 1998, 618–628.

[LW98] K. Lodaya, P. Weil, Kleene iteration for parallelism. In: proc. *FST & TCS'98*, LNCS **1530**, Springer-Verlag, 1998, 355–366.

[LW00] K. Lodaya, P. Weil, Series-parallel languages and the bounded-width property. *Theoret. Comput. Sci.* **237** (2000), 347–380.

[LW01] K. Lodaya, P. Weil, Rationality in algebras with a series operation. *Inform. and Comput.* **171** (2001), 269–293.

[Ném04] Z. L. Németh, A hierarchy theorem for regular languages over free bisemigroups. *Acta Cybern.* **16** (2004), 567–577.

[Ném05] Z. L. Németh, Automata on infinite biposets. In: proc. *Automata and Formal Languages, 11th Int. Conf., AFL'05, Dobogókő, Hungary, May 17–20, 2005*, Inst. of Informatics, Univ. of Szeged, Szeged, 2005, 213–226.

[Ném06] Z. L. Németh, Automata on infinite biposets. *Acta Cybern.* **18** (2006), 765–797.

[Ném07] Z. L. Németh, On the regularity of binoid languages: a comparative approach. In: preproc. *1st International Conference on Language and Automata Theory and Applications, LATA'07*, March 29 – April 4, 2007, Tarragona, Spain.

[Nie01] M. Nielsen, Modelling with partial orders – why and why not? In: proc. *ICALP'01*, LNCS **2076**, Springer-Verlag, 2001, 61–63.

[PP93] D. Perrin, J.-E. Pin, Semigroups and automata on infinite words. In: *Semigroups, Formal Languages and Groups (York, 1993)*, NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci., **466**, Kluwer Acad. Publ., 1995, 49–72.

[PP04] D. Perrin, J.-E. Pin, *Infinite words*. Pure and Applied Mathematics, Vol. **141**, Elsevier, 2004.

[Pin86] J.-E. Pin, *Varieties of formal languages*. Plenum Publishing Corp., New York, 1986.

[Pra86] V. R. Pratt, Modelling concurrency with partial orders. *International Journal of Parallel Programming* **15** (1986), 33–71.

[RS04] N. Robertson, P.D. Seymour, Graph minors. XX. Wagner's conjecture. *J. Combin. Theory*, Ser. B, **92** (2004), 325–357.

[RS97] G. Rozenberg, A. Salomaa (eds.), *Handbook of formal languages.* Vols. 1–3., Springer-Verlag, Berlin, 1997.

[Sch65] M.-P. Schützenberger, On finite monoids having only trivial subgroups. *Inform. and Control* **8** (1965), 190–194.

[Sta99] R.P. Stanley, *Enumerative combinatorics.* Vol. 2, Cambridge University Press, 1999, 219–229.

[Ste98] M. Steinby, General varieties of tree languages. *Theoret. Comput. Sci.* **205** (1998), 1–43.

[Str94] H. Straubing, *Automata, formal logic and circuit complexity.* Birkhäuser, Boston, 1994.

[TW68] J. W. Thatcher, J. B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. Systems Theory* **2** (1968), 57–81.

[Tho90] W. Thomas, Logical definability of trace languages. In: proc. *ASMICS-Workshop "Free Partially Commutative Monoids"*, V. Diekert (ed.), Rep. TUM-I9002, TU München, 1990, 172–182.

[Val78] J. Valdes, Parsing flowcharts and series-parallel graphs. PhD thesis, Stanford University, 1978, STAN-CS-78-682.

[VTL82] J. Valdes, R. E. Tarjan, E. L. Lawler, The recognition of series-parallel digraphs. *SIAM J. Comput.* **11** (1982), 298–313.

[Wei04a] P. Weil, Algebraic recognizability of languages. In: proc. *MFCS'04*, LNCS **3153**, Springer-Verlag, 2004, 149–175.

[Weil04b] P. Weil, On the logical definability of certain graph and poset languages. *J. Autom. Lang. Comb.* **9** (2004), 147–165.

[WikA] WIKIPEDIA CONTRIBUTORS, "Catalan numbers". *Wikipedia, The Free Encyclopedia*, `http://en.wikipedia.org/w/index.php?title=Catalan-numbers&oldid=61560743`, (accessed July 5, 2006).

[WikB] WIKIPEDIA CONTRIBUTORS, "Robertson–Seymour theorem". *Wikipedia, The Free Encyclopedia*, `http://en.wikipedia.org/w/index.php?title=Robertson-Seymour_theorem&oldid=50432779`, (accessed July 27, 2006).

[Wil94] TH. WILKE, An algebraic theory for regular languages of finite and infinite words. *Internat. J. Algebra Comput.* **3** (1993), 447–489.

[Wil97] TH. WILKE, Star-free picture expressions are strictly weaker than first-order logic. In: proc. *ICALP'97*, LNCS **1256**, Springer-Verlag, 1997, 347–357.

Appendix

Here we will supply a detailed description of our verification algorithm used to check the encoded runs in the proof of Theorem 4.18. In addition, we will also briefly describe how to build the formula ψ_3 which realizes the algorithm.

Now suppose that $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ is a parenthesizing automaton, $i \in I$ and $f \in F$. Recall that $\mathrm{SOVar}_{\mathcal{A}} = \{ X_{s_i} \mid s_i \in S \} \cup \{ Z_{\langle i \rangle_i} \mid \langle i, \rangle_i \in \Omega \}$. Let $P = (P, <_h, <_v, \lambda) \in \mathrm{SPB}(\Sigma)$ denote an sp-biposet, and assume that η is an evaluation of the monadic second order variables, that is,

$$\eta : \mathrm{SOVar}_{\mathcal{A}} \to \mathcal{P}(P),$$

where $\mathcal{P}(P)$ denotes the power-set of P. Moreover, assume that P with η satisfies formulas ψ_1 and ψ_2 on page 104.

The following algorithm determines whether η encodes a direct run of $\mathcal{A}$ on P that starts from i and ends in f. For the sake of simplicity, we shall write X_j instead of $\eta(X_j)$. Moreover, in the names of the procedure calls below, "Clan" always means a proper prime clan of P.

Unfortunately, in the definition of function NextState a difficulty arises. As $\mathcal{A}$ is nondeterministic, for a given position x and $s \in S$, there can be more than one t such that $(s, \lambda(x), t) \in \delta$ holds. But when we convert our algorithm into an MSO-formula, we need only test whether $\textsc{NextState}(x) = t$ holds, which resolves the problem.

The pseudocode in Lines 1–10 checks that the run starts from i and ends in f. The code in Lines 11–22 verifies the correctness of the labeling transitions, while Lines 23–49 checks the parenthesizing transitions. The proof of correctness of this algorithm is omitted, but Figures 5.1–5.4 should help the reader to establish it.

The computations which are not represented in the figures are either straightforward or symmetrical to the depicted cases.

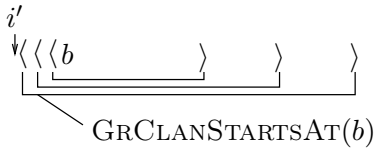

Figure 5.1: *The computation of i' in Line 3.*

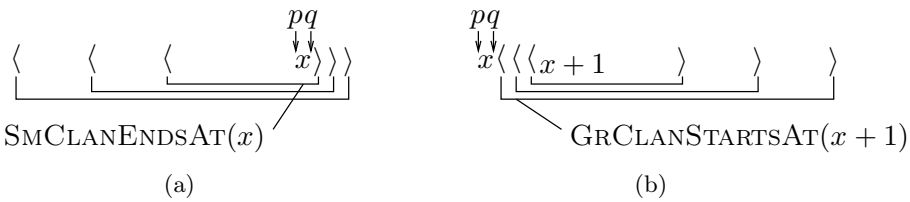

Figure 5.2: *The computation of q in Line 14 (a) and in Line 18 (b).*

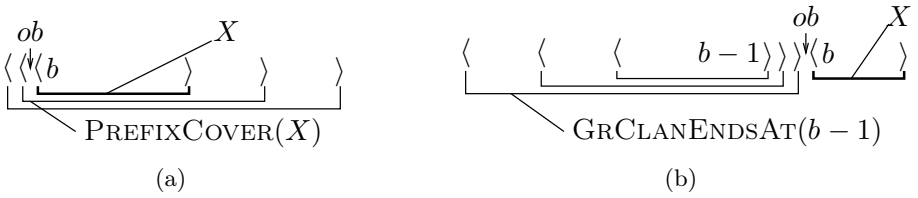

Figure 5.3: *The computation of ob in Line 27 (a) and in Line 31 (b).*

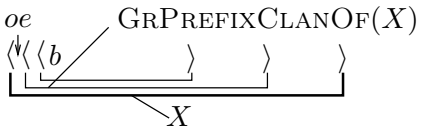

Figure 5.4: *The computation of oe in Line 34.*

Algorithm CORRECT-RUN$(\mathcal{A}, i, f, P, \eta)$

```
 1  b ← FirstOf(P)
 2  if IsClanStartsAt(b)
 3     then i' ← StartOfOpPar(GrClanStartsAt(b))
 4     else i' ← State(b)
 5  e ← LastOf(P)
 6  if IsClanEndsAt(e)
 7     then f' ← EndOfClPar(GrClanEndsAt(e))
 8     else f' ← NextState(e)
 9  if i ≠ i' or f ≠ f'
10     then return 'no'
11  for all x ∈ P
12  do p ← State(x)
13     if IsClanEndsAt(x)
14        then q ← StartOfClPar(SmClanEndsAt(x))
15        else if IsLastPosition(x)
16                then q ← f
17                else if IsClanStartsAt(x + 1)
18                        then q ← StartOfOpPar(GrClanStartsAt(x + 1))
19                        else q ← State(x + 1)
20     σ ← λ(x)
21     if not (p, σ, q) ∈ δ
22        then return 'no'
23  for all proper prime clans X ⊆ P
24  do b ← FirstOf(X)
25     e ← LastOf(X)
26     if IsPrefixOfClan(X)
27        then ob ← EndOfOpPar(PrefixCover(X))
28        else if IsFirstPosition(b)
29                then ob ← i
30                else if IsClanEndsAt(b − 1)
31                        then ob ← EndOfClPar(GrClanEndsAt(b − 1))
32                        else ob ← NextState(b − 1)
33     if IsPrefixClanIn(X)
34        then oe ← StartOfOpPar(GrPrefixClanOf(X))
35        else oe ← State(b)
36     if IsSuffixClanIn(X)
37        then cb ← EndOfClPar(GrSuffixClanOf(X))
38        else cb ← NextState(e)
39     if IsSuffixOfClan(X)
40        then ce ← StartOfClPar(SuffixCover(X))
41        else if IsLastPosition(e)
42                then ce ← f
43                else if IsClanStartsAt(e + 1)
44                        then ce ← StartOfOpPar(GrClanStartsAt(e + 1))
45                        else ce ← State(e + 1)
46     k ← IndexOfParUsedAround(X)
47     if not(ob, ⟨k, oe), (cb, ⟩k, ce) ∈ γ
48        then return 'no'
49  return 'yes'
```

The input-output specifications of the predicates and functions used in the algorithm are the following:

IsFirstPosition(x) / IsLastPosition(x)
input: a position $x \in P$;
output: 'yes' if x is the first/last position of P;
'no' otherwise.

IsClanStartsAt(x) / IsClanEndsAt(x)
input: a position $x \in P$;
output: 'yes' if there is a proper prime clan $X \subseteq P$ whose first/last position is x;
'no' otherwise.

IsPrefixOfClan(X) / IsSuffixOfClan(X)
input: a proper prime clan $X \subseteq P$;
output: 'yes' if there is a proper prime clan Y such that X is a prefix/suffix of Y;
'no' otherwise.

IsPrefixClanIn(X) / IsSuffixClanIn(X)
input: a proper prime clan $X \subseteq P$;
output: 'yes' if there is a proper prime clan Z such that Z is a prefix/suffix of X;
'no' otherwise.

State(x)
input: a position $x \in P$;
output: a state $s \in S$ in which $\mathcal{A}$ reads position x, i.e. $x \in X_s$.

NextState(x)
input: a position $x \in P$;
output: a state $t \in S$ which $\mathcal{A}$ arrives at after reading the position x, i.e. $x \in X_s$ and $(s, \lambda(x), t) \in \delta$.

FirstOf(X) / LastOf(X)
input: a proper prime clan $X \subseteq P$;
output: the first/last position of X.

StartOfOpPar(X) / EndOfOpPar(X)
input: a proper prime clan $X \subseteq P$;

output: a state $s \in S$ such that s is the source/target of an opening parenthesizing transition $(s, \langle_j, t)$ / $(r, \langle_j, s) \in \gamma$, and this transition was used immediately before X, i.e. the designated position of X is in $Z_{\langle_j\rangle_j}$.

StartOfClPar(X) / EndOfClPar(X)

input: a proper prime clan $X \subseteq P$;

output: a state $s \in S$ such that s is the source/target of a closing parenthesizing transition $(s, \rangle_j, t)$ / $(r, \rangle_j, s) \in \gamma$, and this transition was used immediately after X, i.e. the designated position of X is in $Z_{\langle_j\rangle_j}$.

SmClanEndsAt(x) / GrClanEndsAt(x)

input: a position $x \in P$;

output: the smallest/greatest proper prime clan of P that ends at position x.

GrClanStartsAt(x)

input: a position $x \in P$;

output: the greatest proper prime clan of P that starts at position x.

GrPrefixClanOf(X) / GrSuffixClanOf(X)

input: a proper prime clan $X \subseteq P$;

output: the greatest proper prime clan $Y \subseteq X$ that is a proper prefix/suffix of X.

PrefixCover(X) / SuffixCover(X)

input: a proper prime clan $X \subseteq P$;

output: the smallest proper prime clan Y for which is X is a proper prefix/suffix of Y.

IndexOfParUsedAround(X)

input: a proper prime clan X;

output: an index k for which the parentheses $\langle_k, \rangle_k$ were used before and after X in the encoded run, i.e. the designated position of X is in $Z_{\langle_k\rangle_k}$.

Finally, we will outline the transformation of the algorithm into the formula ψ_3. The following observations lead to this transformation.

1. All predicates of the algorithm can be expressed by MSO-formulas. For example, IsPrefixOfClan(X) can be formulated as

$$\exists Y\big(\,\mathrm{PPC}(Y) \wedge \mathrm{Prefix}(X,Y)\,\big)$$

2. For any function $f(x_1,\ldots,x_l)$ of the algorithm and for any element c in the range of f, the fact $f(x_1,\ldots,x_l) = c$ can also be expressed by an MSO-formula. For example, for any state s in S, StartOfOpPar(X) $= s$ can be written as

$$\bigvee_{j\in J} \exists\, z\big(\,\mathrm{Dp}(z,X) \wedge Z_{\langle j\rangle_j}(z)\,\big),$$

where $J = \{\, j \mid \exists\, t \in S, (s, \langle_j, t) \in \gamma \,\}$ is a finite set.

3. The variables whose values are not positions or sets of positions of P all take their values from a finite set. Namely, i, f, i', f', p, q, ob, oe, cb, ce take values from S, σ from Σ, and k is an index of a parenthesis in Ω.

4. The composition of functions can be handled with the aid of auxiliary variables. For example, StartOfOpPar(GrClanStartsAt(b)) $= s$ can be expressed as

$$\exists\, Z\big(\,\textsc{GrClanStartsAt}(b) = Z \wedge \textsc{StartOfOpPar}(Z) = s\,\big)$$

5. Assignments like $y \leftarrow f(x_1,\ldots,x_l)$ can be treated as follows. We can consider all possible values c in the range of y in advance, and at the points of the assignments we can test whether $f(x_1,\ldots,x_l) = c$ holds. If the range of y is P or the power-set $\mathcal{P}(P)$, i.e. y is a 'standard' first or second order variable, then existential quantification can be used. On the other hand, if y is not 'standard', then it has a finite range by point 3. Hence we can use a disjunction over this finite range. For example, we can begin the formula realizing Lines 12–22 with

$$\bigvee_{p\in S}\bigvee_{q\in S}\bigvee_{\sigma\in\Sigma}\cdots$$

6. The control flow of the algorithm can easily be formulated in our logic framework. For the sequential executions conjunctions, for "for all" loops universal quantifications, and for the conditional statements implications and negations can be used.