

Operations on Graphs, Arrays and Automata

மீனாட்சி பரமசிவன்

(Meenakshi Paramasivan)



 **Universität Trier**

September 2017

Operations on Graphs, Arrays and Automata

Dissertation

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
am Fachbereich IV der Universität Trier

Vorgelegt von
M.Phil.-Mathematiker
Meenakshi Paramasivan
aus Chennai, India

Erster Gutachter: Prof. Dr. H. Fernau, Universität Trier
Zweiter Gutachter: Prof. Dr. R. Freund, Universität Wien

Tag der mündlichen Prüfung: 25. September 2017
„Gedruckt mit Unterstützung des Deutschen Akademischen Austauschdienstes“

I am Alpha and Omega, the beginning and the end, the first and the last.
Revelation 22:13

ACKNOWLEDGMENTS

I would like to thank my doctoral father (supervisor), Prof. Dr. Henning Fernau, in making my years of graduation fruitful, enjoyable, and memorable. When I joined the program, all I had was interest in theoretical research in Germany. Henning has provided space to explore as a researcher. I benefited by numerous discussions that I had with him. I owe my interest in formal languages and automata theory to him. His patient guidance, support, and encouragement have led me towards the completion of this thesis. His multi-area research personality, had a great influence on me. Special thanks to him and his wife for the wonderful swimming sessions. For all the things I learned from him, technical and otherwise, I will forever be grateful.

I would like to thank my second doctoral supervisor, Prof. Dr. Rudolf Freund with whom I had useful discussions in connecting the research topics in this thesis to further develop in a simple and unique way, his specific and different thinking made always the discussions to think out of the box/border.

I would like to thank Dr. T. Robinson and Dr. D.G. Thomas for sharing their ideas, for getting me interested in problems related to array grammars, and for many collaborations. A special thanks to Dr. N. G. David for being my mentor. On the most trying days to get through the DAAD Scholarship, his faith in me has helped me to stay for DAAD Scholarship. I would like to thank all teachers from Madras Christian College, especially, Dr. V. Rajkumar Dare and Dr. M. K. Viswanath for their motivation to study further.

I would also like to thank the people with whom I have had useful discussion on technical topics related to this thesis or otherwise. These include Dr. Daniel Meister (discussions on counter automata), Dr. Markus L. Schmid (for clearing my stupid questions always and helping me with latex learning by his specific comments to type in latex), Franz Brauße (Graph Theory sessions) and S. James Immanuel (for discussions on Picture Languages).

I also thank all the Computer Science faculty at University of Trier, who have been responsible for imparting to me their knowledge on various aspects of the computer science through numerous seminars. I would like to thank the university for its financial support and the wonderful infrastructure and working environment that it provides to its members. A special thanks to Heike Bewen and Nicole Trouet-Schartz.

A special thanks to Sagaya Mary (maths teacher in Christ King Girls' Hr. Sec. School), Parvathi (maths teacher for X standard in the year 2002 in school), Thiyagu (tution class - maths teacher for XII - Arul Institute), Mumtaj Fathima (my only math-companion from X standard until today), Kannan Babu, Kalavathi N and Ramya Jose (MCC friends) and also to Kumaran Sivagnanam and James

Babu (from ABN-AMRO, RBS) for their timely supportive words to continue my education. A big thanks to Else Schneider for her friendly and family support to stay responsible in Trier. A warm thanks to all my friends for making my social life enjoyable.

Finally, most importantly, I owe my thanks to my grand parents especially to my ammachi Sagayavalli Perumal, my father Paramasivan Pandian, my mother Sankareswari Paramasivan, sister Sagayavalli Paramasivan, my husband Magesh Kumar and my daughter Naphtali (Tharshanaa) who have filled my life with love, joy and have always had confidence in me, even when I myself did not. With their encouragement and support only, taking up research was possible in a foreign country. Words are not enough to express my gratitude for what they have done for me.

My hearty thanks to the Holy Spirit who is the Comforter (counselor, helper, advocate, intercessor, strengthener and standby), whom the Father has sent in the name of Jesus Christ (John 14:26), and my hearty thanks to K. Antony Sharmila, Pastor. Danial Jebanesan Ramanathan, his wife Sulatha D J and Pastor. Mike Williams.

Last but not least my sincere thanks to Christ King Girls' Hr. Sec. School, Madras Christian College, East Tambaram for the Education provided and to Deutscher Akademischer Austauschdienst (DAAD), Ada-Lovelace-Projekt Trier for the funding provided and to Mensa Trier Petrisberg for its delicious and healthy food and to Liberty Christian Centre Selaiyur and Gemeinde des lebendigen Gottes Trier e.V. for many reasons.

Abstract

Automata theory is the study of abstract machines. It is a theory in theoretical computer science and discrete mathematics (a subject of study in mathematics and computer science). The word automata (the plural of automaton) comes from a Greek word which means “self-acting”. Automata theory is closely related to formal language theory [99, 101]. The theory of formal languages constitutes the backbone of the field of science now generally known as theoretical computer science. This thesis aims to introduce a few types of automata and studies the class of languages recognized by them.

Chapter 1 is the road map with introduction and preliminaries. In Chapter 2 we consider few formal languages associated to graphs that has Eulerian trails. We place few languages in the Chomsky hierarchy that has some other properties together with the Eulerian property.

In Chapter 3 we consider jumping finite automata, i. e., finite automata in which input head after reading and consuming a symbol, can jump to an arbitrary position of the remaining input. We characterize the class of languages described by jumping finite automata in terms of special shuffle expressions and survey other equivalent notions from the existing literature. We could also characterize some super classes of this language class.

In Chapter 4 we introduce boustrophedon finite automata, i. e., finite automata working on rectangular shaped arrays (i. e., pictures) in a boustrophedon mode and we also introduce returning finite automata that reads the input, line after line, does not alters the direction like boustrophedon finite automata i. e., reads always from left to right, line after line. We provide close relationships with the well-established class of regular matrix (array) languages. We sketch possible applications to character recognition and kolam patterns.

Chapter 5 deals with general boustrophedon finite automata, general returning finite automata that read with different scanning strategies. We show that all 32 different variants only describe two different classes of array languages. We also introduce Mealy machines working on pictures and show how these can be used in a modular design of picture processing devices.

In Chapter 6 we compare three different types of regular grammars of array languages introduced in the literature, regular matrix grammars, (regular : regular) array grammars, isometric regular array grammars, and variants thereof, focusing on hierarchical questions. We also refine the presentation of (regular : regular) array grammars in order to clarify the interrelations.

In Chapter 7 we provide further directions of research with respect to the study that we have done in each of the chapters.

Contents

1	Road Map	1
1.1	Origin: Overview	1
1.2	Ingredients: Preliminaries	4
1.2.1	Words, Languages and Machines	5
1.2.2	Two Dimensional World	8
1.2.3	Graphs	13
2	Eulerian Trails	22
2.1	Formal Language Questions for Eulerian Trails	22
2.2	Standard PLD	30
2.3	Eulerian Traces	32
3	Jumping Finite Automata	45
3.1	JFA and Shuffle Expressions	45
3.2	Algebraic Properties: Shuffle and Permutation	49
3.3	The Language Class \mathcal{JFA}	54
3.4	The Language Classes \mathcal{GJFA} and \mathcal{SHUF}	60
3.5	Representations and Normal Forms	70
4	Scanning Automata and Grammars	76
4.1	Boustrophedon Finite Automata	77
4.2	Returning Finite Automata	84
4.3	Regular Matrix Languages	95
4.4	Regular Array Grammars	101
4.5	Pumping and Interchange Lemmas	116
4.5.1	Pumping Lemmas	116
4.5.2	Interchange Lemmas	118
4.5.3	Application of Pumping and Interchange Lemmas	119
4.6	Hierarchy Results, Further Automata Models	120
4.6.1	BFA Languages and Regular Matrix Languages	120
4.6.2	3-Way Automata	122

4.6.3	Isometric Array Languages	123
4.7	Closure Properties	124
4.7.1	Set Operations	124
4.7.2	Reflection-like Operations	125
4.7.3	Catenation and Catenation Closure	126
4.8	Possible Applications to Character Recognition	127
4.9	Possible Applications to Kolam Patterns	128
5	Picture Transforming Automata	132
5.1	General Boustrophedon Finite Automata	132
5.2	General Returning Finite Automata	136
5.3	Language Families under the Unary Operators	137
5.4	Picture Transforming Automata	142
6	Regular Grammars for Array Languages	150
6.1	(Regular : Regular) Array Grammars	150
6.2	Regular Grammars, Isometric Array Languages	157
7	Destination: Further Directions	168

List of Figures

1.1	Diagram of graph G	14
1.2	Diagram of graph H	14
1.3	G , $M(G)$ and $A(G)$	16
1.4	Walk, Trail and Path	19
1.5	A connected graph	19
1.6	A disconnected graph with 3 components	19
1.7	Closed trail and Cycle	20
1.8	Graph Representation of Königsberg Bridge Problem	20
1.9	$n = 1$ and $n = 4$	21
1.10	$n = 2$ and $n = 3$	21
2.1	Graph G and $PLD_\phi(G)$ in Example 7	23
2.2	Eulerian trails W_1 and W_2	25
2.3	$W_1 \cdot_{x_0} W_2$	25
2.4	Graphs G , G' , G'' , G'_1 , G'_2	29
2.5	PLDs of Graphs in Fig.2.4	29
2.6	$PLD(G)$ with respect to PLDs ϕ_1 and ϕ_2 in Example 9	31
2.7	Deterministic Blind One-Counter Machine M that accepts ET°	33
2.8	Case 1a and Case 2a	40
2.9	DFA A	43
3.1	Finite Machine M	49
3.2	General Finite Machine M'	49
3.3	An example JFA, final states not specified.	57
3.4	The finite machine of Example 12.	59
3.5	Inclusion diagram of our language families.	63
4.1	How M^- processes an input.	79
4.2	BFA M that accepts the language L_L in Example 16	80
4.3	Example derivation of the BFA M in Example 16	80
4.4	d-BFA M_d	83
4.5	RFA M' that accepts the language L_L in Example 16	84

4.6	RFA M' constructed by Theorem 23 with only useful states	94
4.7	BFA M constructed by Theorem 23 with only useful states	94
4.8	RFA M constructed by Theorem 24 with all reachable states	100
4.9	A sample parallel derivation of the constructed RMG.	100
4.10	How an IRAG can generate L_{\setminus}	105
4.11	How array processing of automata and grammars complement.	108
4.12	BFA M_D that accepts the language $L_{Rect}(G_{d,D})$ in Illustration 8	114
4.13	BFA M_R that accepts the language $L_{Rect}(G_{d,R})$ in Illustration 8	116
4.14	BFA M_L that accepts the language $L_{Rect}(G_{d,L})$ in Illustration 8	117
4.15	BFA M_{LRD} that accepts the language $L_{Rect}(G_d)$ in Illustration 8	117
4.16	Relations between array language families if $ \Sigma > 1$	121
4.17	Relations between isometric array language families	125
4.18	RFA M^F that accepts the language of F tokens, of all sizes and of all proportions	129
4.19	RFA M that accepts the language of P tokens, of all sizes and of all proportions	129
4.20	RFA accepting $L(M_L) \ominus L(M^F)$	129
4.21	RFA accepting transpose of the set of all Aasanapalakai	130
4.22	A sample element in the set of all Aasanapalakai	131
5.1	GRFA M that accepts the language in Example 25.	136
5.2	GRFA R that accepts $L(R) = \{\bullet\}_+^+ \ominus (\{x\} \oplus \{x\}^+) \ominus \{\bullet\}_*^+$	145
5.3	MPM M' designed according to our description.	145
5.4	GRFA R^\dagger that accepts $M'^-(L(R))$	146
5.5	MPM M	147
5.6	GRFA R' that accepts $M^-(L(R^\dagger))$	149
6.1	Derivation tree for F_2	153
6.2	Case (i) (on left) and Case (ii) (on right) for a sample element in $H(\hat{L})$	158
6.3	Semi-holes	159
6.4	How an IRAG can scan BFA pictures, keeping track of the right border	161
6.5	How to scan the picture M_8 (on the left) and M_9 (on the right)	164
6.6	The world of rectangular array language families	167

List of Tables

1.1	Unary Operators	11
1.2	Table of Operators.	12
4.1	Closure properties of the family $\mathcal{L}(\text{BFA})$	127
5.1	Operators/Directions for GBFAs and GRFAs.	138
5.2	Simplifying array languages with MPMs	144

Chapter 1

Road Map

1.1 Origin: Overview

Operations: what does it mean here?

Operation can refer to medical surgery, a military campaign, or mathematical methods such as \div and \times . Operation comes from the Latin word opus (“work”) and can refer to a whole range of practical activities and work. In the driving lessons, we learn the proper operation of a motor vehicle. In computer science: an operation is performed on the basis of an instruction. In this thesis we deal mostly with mathematical methods.

Operations: why and how to study them?

An operation is a calculation from one value or more input values (called operands) to an output value. The number of operands is the arity of the operation. The most commonly studied operations are binary operations of arity 2, such as $+$ and \times , and unary operations of arity 1, such as additive inverse, multiplicative inverse, negation and trigonometric functions. An operation of arity 0 is a constant.

Operations can involve mathematical objects other than numbers. The logical values TRUE and FALSE can be combined using logic operations, such as AND, OR, and NOT. Vectors can be added and as well as subtracted. Operations on functions include composition, convolution. Rotations can be combined using the function composition operation, performing the first rotation and then the second.

Operations on sets include binary operations \cup and \cap and unary operation of complementation. Operations may not be defined for every possible value. For example, in the real numbers one cannot divide by zero or take square roots of

negative numbers. The values for which an operation is defined form a set called its domain. The set which contains the values produced is called the co-domain, but the set of actual values attained by the operation is its range. For example, in the real numbers, the squaring operation only produces non-negative numbers; the co-domain is the set of real numbers but the range is the non-negative numbers.

Operations can involve dissimilar objects. A vector can be multiplied by a scalar to form another vector. And the inner product operation on two vectors produces a scalar. An operation may or may not have certain properties, as it may be associative, commutative, idempotent, and so on. The values combined are called operands, arguments, or inputs, and the value produced is called the value, result, or output. Operations can have fewer or more than two inputs.

An operation is like an operator, but the point of view is different. For instance, one often speaks of “the operation of +” or “+ operation” when focusing on the operands and result, but one says “+ operator” (rarely “operator of +”) when focusing on the process, or from the more abstract viewpoint, the function $+ : M \times M \rightarrow M$.

Operations: what we study here?

We study the operations on Graphs, Arrays and Automata, now we will see a brief and rough introduction to each of them in the following:

Graphs:

Graph operations produce new graphs from initial ones. They may be separated into the following categories: Unary operations create a new graph from initial one. Elementary operations or editing operations create a new graph from initial one by a simple local change, such as addition or deletion of a vertex or of an edge, merging and splitting of vertices, edge contraction, etc.

Advanced operations create a new graph from one initial one by a complex changes, for instance; transpose graph, complement graph, line graph, graph rewriting and dual graph etc. Binary operations create a new graph from two initial ones G_1 and G_2 such as: graph union, graph intersection, and graph join etc.

Arrays:

Array operations are also is of two types, unary and binary. Unary operators that we have discussed in this thesis form a dihedral group. Binary operators are row (column) catenation of two arrays. Array operations are helpful to find results on closure properties of array languages, to find the connection to character recognition and to apply to make Kolam patterns.

Automata:

We can analyze, modify and combine automata. By analyzing we mean to find the unreachable states which are unnecessary (not useful), also transitions associated with those states. This in short can be said as finding the accessible part of the automata by deleting those states. By modifying as described above we obtain an automaton with only useful states for which we can find the complement when it is a complete one. For combining automata, two operations are normally considered, one is parallel composition and another one is product which we will see often. We will also see automata with input and output.

Thesis: Contribution

Some major results of this thesis have been published by the author in some of the conference proceedings or journals. Now we see very briefly how those results, presented in the forthcoming chapters, correspond to those articles:

Chapter 2 has few results been originally presented in [28]. Section 3.3 and Section 3.4 are presented in [29] and its journal version [33]. Chapter 4, contains work published in [30] and [34] its journal version [31]. Most of the results of Chapter 5 and Chapter 6 are presented in [35] and [36], respectively.

Complexity results from the articles mentioned above, were not the original contribution from the author, instead those were contributed by other co-authors so those results are excluded in this thesis.

In this thesis some of the sections has well explained results with proofs, lots of explanations whereas some are not, as those are not published with proofs are available with detailed proofs for instance; Theorem 25 has an alternative proof in this thesis compared to the one in [31] and some of the inductive proofs and illustrations.

1.2 Ingredients: Preliminaries

In this section we give some notations and then we give an overview of some standard definitions concerning words, languages and machines. We follow the terminologies, basic notions of formal languages and automata theory as in [50]. We give the same for two dimensional words, languages in the second subsection and also for graphs in the third subsection.

Let $\mathbb{N} := \{1, 2, 3, \dots\}$ be the set of natural numbers and let $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. Let S be a set. Then $Card(S)$ is the number of its elements. The *empty set* is denoted by \emptyset . If M, N are subsets of S , then we write $M \subseteq N$ if and only if $x \in M \Rightarrow x \in N$, and $M \subsetneq N$ if and only if $M \subseteq N$ and $M \neq N$. And

$$M \cup N = \{x \in S \mid x \in M \text{ or } x \in N\}.$$

$$M \cap N = \{x \in S \mid x \in M \text{ and } x \in N\}.$$

$$M \setminus N = \{x \in S \mid x \in M \text{ and } x \notin N\}.$$

A *singleton* is a subset of S consisting of just one element. If no confusion can arise, we shall not distinguish elements of S from singletons. The set of all subsets of S , i. e., the powerset of S , is denoted by $\mathcal{P}(S)$ or 2^S . With the preceding convention, $S \subseteq \mathcal{P}(S)$. For any sets A and B , the set of all total functions $f : A \rightarrow B$ is denoted B^A . The *domain* $dom(f)$ of a partial function $f : S \rightarrow T$ is the set of elements $x \in S$ for which $f(x)$ is defined. f can be viewed as a (total) function from S into $\mathcal{P}(T)$, and with the convention $T \subseteq \mathcal{P}(T)$, as a total function from S into $T \cup \{\emptyset\}$. Then $dom(f) = \{x \in S \mid f(x) \neq \emptyset\}$. The hull operator (closure operator) on a set S is a function $H : 2^S \rightarrow 2^S$ from the power set of S to itself which satisfies the following conditions for all $A, B \subseteq S$:

$$A \subseteq H(A) \text{ (H is extensive)}$$

$$A \subseteq B \Rightarrow H(A) \subseteq H(B) \text{ (H is increasing)}$$

$$H(H(A)) = H(A) \text{ (H is idempotent)}.$$

For $n \geq 1$, let \mathbb{N}^n be the n -fold Cartesian product of \mathbb{N} with itself. For $x, y \in \mathbb{N}^n$, i. e., $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$, let $x + y = (x_1 + y_1, \dots, x_n + y_n)$ and for $c \in \mathbb{N}$, let $cx = (cx_1, \dots, cx_n)$.

A *semigroup* consists of a set G and a binary operation on G , denoted by $*$, and which is assumed to be *associative*: For any $a, b, c \in G$, $a*(b*c) = (a*b)*c$. An *identity* element or a *unit* is an element $e \in G$ such that $e*a = a*e = a$ for all $a \in G$. A semigroup which has a identity element is a *monoid*. The identity element of a monoid is unique.

Given two subsets A, B of a monoid M , the product AB is defined by

$$AB = \{c \in M \mid \exists a \in A, \exists b \in B : c = ab\} \quad (1.1)$$

A *semiring* consists of a set R and of two binary operations, called addition and multiplication, noted $+$ and \cdot , and satisfying the following conditions:

- R is a commutative monoid for the addition ($a + b = b + a$ for all $a, b \in R$) with identity element 0 ;
- R is a monoid for multiplication;
- Multiplication is distributive with respect to the addition: for all $a, b, c \in R$

$$a \cdot (b + c) = a \cdot b + a \cdot c;$$

$$(a + b) \cdot c = a \cdot c + b \cdot c;$$
- For all $a \in R$, $0 \cdot a = a \cdot 0 = 0$.

If R is a monoid, then $\mathcal{P}(R)$ is a semiring with set union for addition and the multiplication (1.1).

1.2.1 Words, Languages and Machines

An *alphabet* Σ is a finite, non-empty set of symbols. A *word (string)* (of length $k > 0$) over an alphabet Σ is a finite sequence of symbols (elements) from Σ . The *empty string* or *empty word* is the string with zero occurrences of symbols, denoted by ε . The *length* of a word w , is the number of symbols that occur in w , denoted by $|w|$. The number of occurrences of a in w is denoted by $|w|_a$. For example $|graph| = 5$ and $|graph|_p = 1$. Note that $|\varepsilon| = 0$.

If Σ is an alphabet, then Σ^k is defined to be the set of strings of length k . Note that $\Sigma^0 = \{\varepsilon\}$. The set of all words including ε over an alphabet Σ is denoted by Σ^* . The set of all non-empty words over Σ is denoted by $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. Clearly, $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$. Put another way, $\Sigma^+ = \bigcup_{n=1}^{\infty} \Sigma^n$ and $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$.

Let u and v be words. Then uv denotes the *concatenation* of u and v , we write $u \cdot v$ or simply uv . More precisely, if $u = a_1a_2 \cdots a_i$ and $v = b_1b_2 \cdots b_j$, then $|uv| = i + j$, $uv = a_1a_2 \cdots a_ib_1b_2 \cdots b_j$. For any word w , $\varepsilon w = w\varepsilon = w$. That is, ε is the *identity for concatenation*. Note that (Σ^*, \cdot) is a free monoid generated by Σ with identity ε .

We say that a string $v \in \Sigma^*$ is a *factor* of a string $w \in \Sigma^*$ if there are $u_1, u_2 \in \Sigma^*$ such that $w = u_1 \cdot v \cdot u_2$. If u_1 or u_2 is the empty string, then v is a *prefix* (or a *suffix*, respectively) of w .

If Σ is an alphabet, $L \subseteq \Sigma^*$, then L is a *language over Σ* . The *reversal (mirror image)* of $w = a_1a_2 \dots a_n$, denoted by w^R , is defined as $w^R = a_n a_{n-1} \dots a_2 a_1$. Note that $\varepsilon^R = \varepsilon$. If L is a language, then the reversal of L , denoted by L^R , is defined as $L^R = \{w^R : w \in L\}$ is the language consisting of the reversals of all its strings. For instance, if $L = \{001, 10, 111\}$, then $L^R = \{100, 01, 111\}$.

A *deterministic finite automaton*, in short DFA consists of a finite set of states and a set of transitions from state to state that occur on input symbols chosen from an alphabet Σ . For each input symbol there is exactly one transition out of each state (possibly back to the state itself). One state, usually denoted q_0 or s is the initial state, in which the automaton starts. Some states are designated as final or accepting states. A directed graph, called a *transition diagram*, is associated with an DFA as follows. The vertices of the graph correspond to the states of the DFA. If there is a transition from state q to state p on input a , then there is an arc labeled a from state q to state p in the transition diagram. The DFA accepts a string x if the sequence of transitions corresponding to the symbols of x leads from the start state to an accepting state.

We formally denote a DFA by a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of *states*, Σ is a finite *input alphabet*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *final states*, and δ is the transition function mapping $Q \times \Sigma$ to Q . That is, $\delta(q, a)$ is a state for each state q and input symbol a .

The behavior of a DFA on a string can be described by extending the transition function δ to apply to a state and a string rather than a state and a symbol. We define a function $\hat{\delta}$ from $Q \times \Sigma^*$ to Q . The intension is the $\hat{\delta}(q, w)$ is the unique state p such that there is a path in the transition diagram from q to p , labeled w . More formally

1. $\hat{\delta}(q, \varepsilon) = q$, and
2. for all strings x and input symbols a ,

$$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a).$$

A string w is said to be *accepted* by a DFA $M = (Q, \Sigma, \delta, q_0, F)$ if $\hat{\delta}(q_0, w) = p$ for some $p \in F$. The language accepted by M , designated $L(M)$, is the set $\{w \mid \hat{\delta}(q_0, w) \in F\}$. Consider modifying the DFA model to allow zero, one, or

more transitions from a state on the same input symbol. This model is said to be *nondeterministic finite automaton*, in short NFA.

A language is a *regular set* (or just *regular*) if it is the set accepted by some finite automaton. The term “regular” comes from “regular expressions”, another formalism we see next and which defines the same class of languages as the finite automaton’s. We denote this language class by \mathcal{REG} .

Definition 1. *Let Σ be an alphabet. The regular expressions over Σ and the sets that they denote are defined recursively as follows:*

- \emptyset is a regular expression and denotes $\{\}$, i. e., $L(\emptyset) = \{\}$.
- ε is a regular expression and denotes the set $\{\varepsilon\}$, i. e., $L(\varepsilon) = \{\varepsilon\}$.
- For each $a \in \Sigma$, \mathbf{a}^1 is a regular expression and denotes the set $\{a\}$, i. e., $L(\mathbf{a}) = \{a\}$.
- If r, s are regular expressions denoting the languages R, S respectively, then $(r + s)$, (rs) and (r^*) are regular expressions that denote the sets $R \cup S$, RS , and R^* , respectively.

In writing regular expressions we can omit parentheses if we assume that $*$ has higher precedence than concatenation or $+$, and that concatenation has higher precedence than $+$. For example, $((0(1^*)) + 0)$ may be written $01^* + 0$. We may also abbreviate the expression rr^* by r^+ .

Remark 1. *When necessary to distinguish between a regular expression r and the language denoted by r , we use $L(r)$ for the latter. When no confusion is possible we use r for both the regular expression and the language denoted by the regular expression.*

Let us now recall the definitions of (multi) counter machines [37, 47]. Let \mathbb{Z} be the set of integers (positive, negative and zero) and $\vec{0}$ be the multidimensional all-zero-vector. $sgn(x)$ is the sign of integer x , i. e., $sgn(x) = -1$ if $x < 0$. $sgn(x) = 0$ if $x = 0$. $sgn(x) = 1$ if $x > 0$.

Definition 2. *A k -counter machine $M = (Q, \Sigma, \delta, q_0, F, k)$ consists of a finite set Q of states, a designated initial state q_0 , a designated subset F of final or accepting states, a finite input alphabet Σ , $k \geq 1$ and a finite transition relation*

$$\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \{-1, 0, 1\}^k \times Q \times \{-1, 0, 1\}^k.$$

¹When one symbol is part of a regular expression, it is written in boldface. However we view \mathbf{a} and a as the same symbol.

A configuration c of M is a member of $Q \times \Sigma^* \times \mathbb{Z}^k$. The set of configurations is denoted by $C(M)$. Especially, $c_0(w) = (q_0, w, \vec{0})$ is the initial configuration for w and $C_F = F \times \{(\varepsilon, \vec{0})\}$ is the set of final configurations.

If $(q, a, u_1, \dots, u_k, q', v_1, \dots, v_k) \in \delta$, (q, aw, y_1, \dots, y_k) is a configuration of M with $u_i = \text{sgn}(y_i)$ for $1 \leq i \leq k$, then we write

$$(q, aw, y_1, \dots, y_k) \vdash_M (q', w, y_1 + v_1, \dots, y_k + v_k).$$

If $a = \varepsilon$, this is an ε -move. \vdash_M is a relation on $Q \times \Sigma^* \times \mathbb{Z}^k$. Its reflexive transitive closure is denoted by \vdash_M^* . The language accepted by M is

$$L(M) = \{w \in \Sigma^* : \exists c_F \in C_F (c_0(w) \vdash_M^* c_F)\}.$$

We restrict our machines to be “blind” by forcing identical action for all counter configurations, and restrict them to be “partially blind”, by not allowing transitions for negative counters and by forcing other transitions to ignore counter contents.

Definition 3. A k -counter machine $M = (Q, \Sigma, \delta, q_0, F, k)$ is blind if for each $q, q' \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, and for all $u_i, v_i, x_i \in \{0, 1, -1\}$

$$(q, a, u_1, \dots, u_k, q', v_1, \dots, v_k) \in \delta \iff (q, a, x_1, \dots, x_k, q', v_1, \dots, v_k) \in \delta.$$

In other words, a blind counter machine is unable to check the signs of its counters during a computation. Only at the end, the acceptance condition checks whether all counters are zero. A partially blind multi-counter machine may be viewed as a blind multicounter machine which gets stuck when one of its counters decreases below zero. Formal definitions of (partially) blind (multi) counter machines are given in [37, 47].

Definition 4. A k -counter machine $M = (Q, \Sigma, \delta, q_0, F, k)$ is deterministic if for each $q \in Q$, $a \in \Sigma$ and for all $u_i \in \{-1, 0, 1\}$ it is true that

$$\begin{aligned} & |\{(q, a, u_1, \dots, u_k, q', v_1, \dots, v_k) \in \delta : q' \in Q, v_i \in \{-1, 0, 1\}\}| \\ & + |\{(q, \varepsilon, u_1, \dots, u_k, q', v_1, \dots, v_k) \in \delta : q' \in Q, v_i \in \{-1, 0, 1\}\}| \leq 1. \end{aligned}$$

1.2.2 Two Dimensional World

Let us now give an overview of the standard definitions and notations regarding two-dimensional words and languages. We follow the terminologies, notations as in [106], [41].

A *two-dimensional word* (also called as *picture*, *matrix* or an *array*) over Σ is a tuple

$$W := ((a_{1,1}, a_{1,2}, \dots, a_{1,n}), (a_{2,1}, a_{2,2}, \dots, a_{2,n}), \dots, (a_{m,1}, a_{m,2}, \dots, a_{m,n})),$$

where $m, n \in \mathbb{N}$ and, for every i , $1 \leq i \leq m$, and j , $1 \leq j \leq n$, $a_{i,j} \in \Sigma$. We define the *number of columns* (or *width*) and *number of rows* (or *height*) of W by $|W|_c := n$ and $|W|_r := m$, respectively. For the sake of convenience, we also denote W by $[a_{i,j}]_{m,n}$ or by a matrix in a more pictorial form. If we want to refer to the j^{th} symbol in row i of the picture W , then we use $W[i, j] = a_{i,j}$.

By Σ_+^+ , we denote the set of all (non-empty) pictures over Σ , every subset $L \subseteq \Sigma_+^+$ is a *picture language*. $\bar{L} = \Sigma_+^+ \setminus L$ is the *complement* of the picture language L .

Let $W := [a_{i,j}]_{m,n}$ and $W' := [b_{i,j}]_{m',n'}$ be two non-empty pictures over Σ . The *column concatenation* of W and W' , denoted by $W \oplus W'$, is undefined if $m \neq m'$ and is the picture

$$\begin{array}{cccccccc} a_{1,1} & a_{1,2} & \dots & a_{1,n} & b_{1,1} & b_{1,2} & \dots & b_{1,n'} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} & b_{2,1} & b_{2,2} & \dots & b_{2,n'} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} & b_{m',1} & b_{m',2} & \dots & b_{m',n'} \end{array}$$

otherwise. The *row concatenation* of W and W' , denoted by $W \ominus W'$, is undefined if $n \neq n'$ and is the picture

$$\begin{array}{cccc} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \\ b_{1,1} & b_{1,2} & \dots & b_{1,n'} \\ b_{2,1} & b_{2,2} & \dots & b_{2,n'} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m',1} & b_{m',2} & \dots & b_{m',n'} \end{array}$$

otherwise. In order to denote that, e. g., $U \oplus V$ is undefined, we also write $U \oplus V = \text{undef}$.

Example 1. *Let*

$$W_1 := \begin{array}{ccc} a & b & a \\ b & c & a \\ a & b & b \end{array}, \quad W_2 := \begin{array}{cc} b & c \\ b & a \\ c & a \end{array} \quad \text{and} \quad W_3 := \begin{array}{ccc} a & b & c \\ c & b & b \end{array}.$$

Then $W_1 \ominus W_2 = W_1 \oplus W_3 = \text{undef}$, but

$$W_1 \oplus W_2 = \begin{array}{ccccc} a & b & a & b & c \\ b & c & a & b & a \\ a & b & b & c & a \end{array} \quad \text{and} \quad W_1 \ominus W_3 = \begin{array}{ccc} a & b & a \\ b & c & a \\ a & b & b \\ a & b & c \\ c & b & b \end{array}.$$

For a picture W and $k, k' \in \mathbb{N}$, by W^k we denote the k -fold column-concatenation of W , by W_k we denote the k -fold row-concatenation of W , and we write $W_{k'}^k := (W^k)_{k'}$.

The row and column catenation operations can be also viewed as operations on languages. If L_1 and L_2 are two picture languages the *column product* is defined as

$$L_1 \oplus L_2 = \{W \oplus W' : W \in L_1, W' \in L_2\}$$

and the *row product* is defined as

$$L_1 \ominus L_2 = \{W \ominus W' : W \in L_1, W' \in L_2\}$$

Let L be a picture language and $L^{1,\oplus} = L$, $L^{i+1,\oplus} = L^{i,\oplus} \oplus L$ for $i \geq 1$; then

$$L^+ = \bigcup_{i=1}^{\infty} L^{i,\oplus} \text{ (column concatenation plus closure)}$$

Let L be a picture language and $L_{1,\ominus} = L$, $L_{i+1,\ominus} = L_{i,\ominus} \ominus L$ for $i \geq 1$; then

$$L_+ = \bigcup_{i=1}^{\infty} L_{i,\ominus} \text{ (row concatenation plus closure)}$$

Also, we define n -fold iterations (powers) of column catenation as W^n and n -fold iterations (powers) of row catenation as W_n . Accordingly, Σ_m^n is understood, as well as $\Sigma_m^+ = \bigcup_{n \geq 1} \Sigma_m^n$ and similarly Σ_+^n . In this sense, $\Sigma^{++} = \Sigma_+^+$.

Example 2. *One can use the operations given so far to describe array languages by expressions. For instance, $L_0 = \{0\}_+^+$ is the set of all arrays filled with zeros. $L_1 = \{1\}_+ \oplus L_0$ is the language of all arrays over $\{0, 1\}$ whose first column is filled with ones, and all other (non-zero many) positions are filled with zeros. Finally, $L_{\perp} = L_0 \oplus L_1$ is the language of all arrays that contain one column c that is completely filled with ones, but all other positions are filled with zeros; additionally there is at least one column to the left of c and one to the right of c .*

Unary Operations and Connections to Group Theory

As pictures are (also) geometrical objects, several further unary operations can be introduced [106]: *quarter-turn* (rotate clockwise by 90°) Q , *half-turn* (rotate by 180°) H , *anti-quarter-turn* (rotate anti-clockwise by 90° (or rotate clockwise by 270°)) Q^{-1} , *transpose* T (reflection along the main diagonal), *anti-transpose* T' (reflection along the anti-diagonal), R_h (reflection along a horizontal (base) line), R_v (reflection along a vertical line). Together with the *identity* I , these (now eight)

operators form a non-commutative group (with respect to composition), the well-known dihedral group D_4 [6]; see Table 1.1a.

In Table 1.1a, \circ is the function composition. So, if $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ are two functions, then $g \circ f : X \rightarrow Z$ is defined by $(g \circ f)(x) = g(f(x))$ for all $x \in X$. How Table 1.1a works is shown below for $W := [a_{i,j}]_{m,n}$ in the following.

$$(T \circ R_v)(W) = T \begin{pmatrix} a_{1,n} & \dots & a_{1,2} & a_{1,1} \\ a_{2,n} & \dots & a_{2,2} & a_{2,1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,n} & \dots & a_{m,2} & a_{m,1} \end{pmatrix} = \begin{matrix} a_{1,n} & a_{2,n} & \dots & a_{m,n} \\ \vdots & \ddots & \vdots & \vdots \\ a_{1,2} & a_{2,2} & \dots & a_{m,2} \\ a_{1,1} & a_{2,1} & \dots & a_{m,1} \end{matrix} = Q^{-1}(W).$$

Table 1.1: Unary Operators

\circ	I	Q^{-1}	H	Q	R_v	R_h	T	T'
I	I	Q^{-1}	H	Q	R_v	R_h	T	T'
Q^{-1}	Q^{-1}	H	Q	I	T	T'	R_h	R_v
H	H	Q	I	Q^{-1}	R_h	R_v	T'	T
Q	Q	I	Q^{-1}	H	T'	T	R_v	R_h
R_v	R_v	T'	R_h	T	I	H	Q	Q^{-1}
R_h	R_h	T	R_v	T'	H	I	Q^{-1}	Q
T	T	R_v	T'	R_h	Q^{-1}	Q	I	H
T'	T'	R_h	T	R_v	Q	Q^{-1}	H	I

(a) Composition table of unary operators.

	Normal Form
I	$(Q \circ Q) \circ (Q \circ Q)$
Q^{-1}	$Q \circ (Q \circ Q)$
H	$Q \circ Q$
R_v	$Q \circ T$
R_h	$T \circ Q$
T'	$Q \circ (Q \circ T)$

(b) Normal form for the unary operators with Q and T .

Let $\mathcal{O} = \{I, Q^{-1}, H, Q, R_v, R_h, T, T'\}$ be the set of these 8 unary operators comprising D_4 . The operators in D_4 are usually partitioned into the four rotations (including the identity) $\{I, Q^{-1}, H, Q\}$, which form the subgroup D_2 of D_4 , and four reflections $\{R_v, R_h, T, T'\}$.

These operations can be also applied (picture-wise) to picture languages and

(language-wise) to families of picture languages. It is interesting to add the fact that one single rotation Q generates all rotations (as a subgroup of D_4) and that all of D_4 are generated by one rotation Q and one reflection T .

In Table 1.1b we make explicit how any operator can be written using the composition of the two mentioned operators. Table 1.1b can be deduced from Table 1.1a, for instance $Q^{-1} = Q \circ (Q \circ Q)$ since $Q^{-1} = Q \circ H$ and $H = Q \circ Q$. This simple observation helps simplify several of our arguments.

For instance, we can combine sequences of catenation and unary operations from D_4 to obtain Table 1.2, starting out from the four simple observations (1) $Q(W_1 \ominus W_2) = Q(W_2) \oplus Q(W_1)$ (however, mind the sequence of arguments), (2) $Q(W_1 \oplus W_2) = Q(W_2) \ominus Q(W_1)$, (3) $T(W_1 \ominus W_2) = T(W_1) \oplus T(W_2)$ and (4) $T(W_1 \oplus W_2) = T(W_1) \ominus T(W_2)$. For example,

$$\begin{aligned}
H(W_1 \ominus W_2) &= (Q \circ Q)(W_1 \ominus W_2) \text{ (By Table 1.1b)} \\
&= Q(Q(W_1 \ominus W_2)) \text{ (By the definition of } \circ) \\
&= Q(Q(W_2) \oplus Q(W_1)) \text{ (Apply Observation (1))} \\
&= Q(Q(W_1)) \ominus Q(Q(W_2)) \text{ (Apply Observation (2))} \\
&= H(W_1) \ominus H(W_2) \text{ (By Table 1.1b)}
\end{aligned}$$

Table 1.2: Table of Operators.

	$W_1 \ominus W_2$	$W_1 \oplus W_2$
I	$I(W_1) \ominus I(W_2)$	$I(W_1) \oplus I(W_2)$
Q	$Q(W_2) \oplus Q(W_1)$	$Q(W_1) \ominus Q(W_2)$
Q^{-1}	$Q^{-1}(W_1) \oplus Q^{-1}(W_2)$	$Q^{-1}(W_2) \ominus Q^{-1}(W_1)$
H	$H(W_2) \ominus H(W_1)$	$H(W_2) \oplus H(W_1)$
T	$T(W_1) \oplus T(W_2)$	$T(W_1) \ominus T(W_2)$
T'	$T'(W_2) \oplus T'(W_1)$	$T'(W_2) \ominus T'(W_1)$
R_v	$R_v(W_1) \ominus R_v(W_2)$	$R_v(W_2) \oplus R_v(W_1)$
R_h	$R_h(W_2) \ominus R_h(W_1)$	$R_h(W_1) \oplus R_h(W_2)$

Remark 2. Recall that all these reflection operations are self-inverse, i. e.,

$$T \circ T = R_h \circ R_h = R_v \circ R_v = I,$$

where I is the identity on Σ_{\pm}^{\dagger} . (See Table 1.1a).

Example 3. Let us continue with Example 2. $T(L_0) = R_v(L_0) = R_h(L_0) = L_0$.
 $R_v(L_1) \neq R_h(L_1) = L_1 \neq T(L_1)$. $R_v(L_1) = R_h(L_1) = L_1 \neq T(L_1)$.

Having \circ as the function composition, we obtain the following lemma that has two identities, which we will use later.

Lemma 1. $R_h = T \circ R_v \circ T$ and $R_v = T \circ R_h \circ T$.

Proof. Due to Remark 2, it is sufficient to show that $R_h \circ T = T \circ R_v$, as both claimed identities follow from this single identity. Let us first consider $R_h \circ T$:

$$(R_h \circ T)(W) = R_h \begin{pmatrix} a_{1,1} & a_{2,1} & \dots & a_{m,1} \\ a_{1,2} & a_{2,2} & \dots & a_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,n} & a_{2,n} & \dots & a_{m,n} \end{pmatrix} = \begin{pmatrix} a_{1,n} & a_{2,n} & \dots & a_{m,n} \\ \vdots & \ddots & \vdots & \vdots \\ a_{1,2} & a_{2,2} & \dots & a_{m,2} \\ a_{1,1} & a_{2,1} & \dots & a_{m,1} \end{pmatrix} = Q^{-1}(W)$$

and then we already have that $(T \circ R_v)(W) = Q^{-1}(W)$. \square

Remark 3. It is also easy to observe how these operations combine, e. g., $T(W_1 \oplus W_2) = T(W_1) \oplus T(W_2)$ and $T(W_1 \ominus W_2) = T(W_1) \ominus T(W_2)$, or $R_v(W_1 \oplus W_2) = R_v(W_1) \oplus R_v(W_2)$, $R_v(W_1 \ominus W_2) = R_v(W_1) \ominus R_v(W_2)$, and similarly with R_h . Also, we have $T(W^+) = (T(W))_+$, $T(W_+) = (T(W))^+$, $R_v(W^+) = (R_v(W))^+$ and $R_v(W_+) = (R_v(W_+))$, and similarly with R_h .

Example 4. Let us apply Remark 3 to our previous examples.

$$\begin{aligned} R_v(L_1) &= R_v(\{1\}_+ \oplus L_0) = R_v(L_0) \oplus R_v(\{1\}_+) = L_0 \oplus \{1\}_+ \neq L_1. \\ R_h(L_1) &= R_h(\{1\}_+ \oplus L_0) = R_h(\{1\}_+) \oplus R_h(L_0) = \{1\}_+ \oplus L_0 = L_1. \\ T(L_1) &= T(\{1\}_+ \oplus L_0) = T(\{1\}_+) \ominus R_h(L_0) = \{1\}_+^+ \ominus L_0 \neq L_1. \\ R_v(L_1) &= R_v(L_0 \oplus L_1) = R_v(L_1) \oplus R_v(L_0) = (L_0 \oplus \{1\}_+) \oplus L_0 = L_1. \\ R_h(L_1) &= R_h(L_0 \oplus L_1) = R_h(L_0) \oplus R_h(L_1) = L_0 \oplus L_1 = L_1. \\ T(L_1) &= T(L_0 \oplus L_1) = T(L_0) \ominus T(L_1) = L_0 \ominus \{1\}_+^+ \ominus L_0 =: L_-. \end{aligned}$$

We will use later L_+ and L_- frequently as simple (counter-) example languages.

1.2.3 Graphs

Let us now give a brief overview of some standard definitions concerning the graphs. We follow the terminologies, basic notations of Graph Theory as in [10].

Definition 5. A graph $G = (V(G), E(G), \psi_G)$ is an ordered triple, where $V(G)$ is the nonempty finite set of vertices, $E(G)$ is a set of edges disjoint from $V(G)$, and ψ_G is an incidence function that associates each edge of G to an unordered pair of (not necessarily distinct) vertices of G , which we write as words of length two over the alphabet $V(G)$. If e is an edge and u and v are vertices such that $\psi_G(e) = uv$, then e is said to join u and v ; the vertices u and v are called the ends of e .

Example 5. $G = (V(G), E(G), \psi_G)$ where $V(G) = \{v_1, v_2, v_3, v_4, v_5\}$, $E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$ and ψ_G is defined by $\psi_G(e_1) = v_1v_2$, $\psi_G(e_2) = v_2v_3$, $\psi_G(e_3) = v_3v_3$, $\psi_G(e_4) = v_3v_4$, $\psi_G(e_5) = v_2v_4$, $\psi_G(e_6) = v_4v_5$, $\psi_G(e_7) = v_2v_5$, $\psi_G(e_8) = v_2v_5$. The diagram of G is given in Fig. 1.1.

Example 6. $H = (V(H), E(H), \psi_H)$ where $V(H) = \{u, v, w, x, y\}$, $E(H) = \{a, b, c, d, e, f, g, h\}$ and ψ_H is defined by $\psi_H(a) = uv$, $\psi_H(b) = uu$, $\psi_H(c) = vw$, $\psi_H(d) = wx$, $\psi_H(e) = vx$, $\psi_H(f) = wx$, $\psi_H(g) = ux$, $\psi_H(h) = xy$. The diagram of H is in Fig. 1.2.

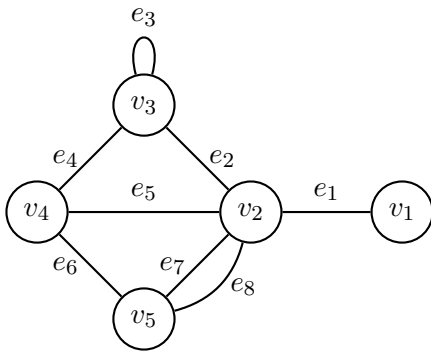


Figure 1.1: Diagram of graph G

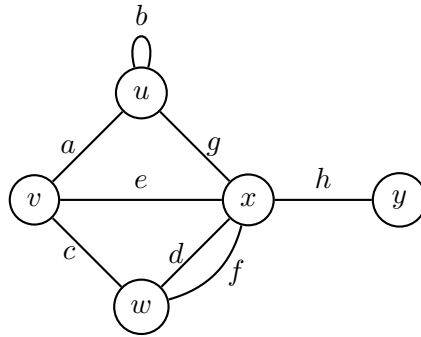


Figure 1.2: Diagram of graph H

Definition 6. The ends of an edge are said to be incident with the edge, and vice versa. Two vertices which are incident with a common edge are adjacent, as are two edges which are incident with a common vertex. An edge with identical ends is called a loop, and an edge with distinct ends a link. Two or more links that join the same pair of vertices are called parallel edges.

For example, e_3 of G (Fig. 1.1) is a loop; all other edges of G are links.

Remark 4. Note that the parallel loops do not get the name of parallel edges.

Definition 7. A graph is simple if it has no loops and no parallel edges.

The graphs of Fig. 1.1 and Fig. 1.2 are not simple. The number of vertices of a graph G is called the *order* of the graph, denoted by $\nu(G)$ and the number of edges of the graph G is called the *size* of the graph, denoted by $\varepsilon(G)$. Hereafter the letter G denotes a graph. Moreover, when just one graph is under discussion, we usually denote this graph by G . We then omit the letter G from graph-theoretic symbols and write, for instance, V, E, ν and ε instead of $V(G), E(G), \nu(G)$ and $\varepsilon(G)$.

Definition 8. Two graphs G and H are identical (written $G = H$) if $V(G) = V(H)$, $E(G) = E(H)$, and $\psi_G = \psi_H$.

Definition 9. Two graphs G and H are said to be isomorphic (written $G \cong H$) if there are bijections $\theta : V(G) \rightarrow V(H)$ and $\phi : E(G) \rightarrow E(H)$ such that $\psi_G(e) = uv$ if and only if $\psi_H(\phi(e)) = \theta(u)\theta(v)$; such a pair (θ, ϕ) of mappings is called an isomorphism between G and H .

To show two graphs are isomorphic, one must indicate an isomorphism between them. Note that G and H in Fig. 1.2 are not identical, but isomorphic.

Definition 10. A simple graph in which each pair of distinct vertices is joined by an edge is called a complete graph.

Up to isomorphism, there is just one complete graph on n vertices; it is denoted by K_n . Here we can note that $\nu(K_n) = n$ and $\varepsilon(K_n) = n(n - 1)/2$.

Definition 11. A graph G is called as empty graph if $\nu(G) = \varepsilon(G) = 0$.

Definition 12. A graph G is called as trivial graph if $\nu(G) = 1$, $\varepsilon(G) = 0$.

Definition 13. A bipartite graph is one whose vertex set can be partitioned into two subsets X and Y , so that each edge has one end in X and one end in Y ; such a partition (X, Y) is called a bipartition of the graph.

Definition 14. A complete bipartite graph is a simple bipartite graph with the bipartition (X, Y) in which each vertex of X is joined to each vertex of Y ; if $|X| = m$ and $|Y| = n$, such a graph is denoted by $K_{m,n}$ where $\nu(K_{m,n}) = m + n$, $\varepsilon(K_{m,n}) = mn$.

Let us denote the vertices of G by v_1, v_2, \dots, v_ν and the edges by $e_1, e_2, \dots, e_\varepsilon$. To any graph G there corresponds a $\nu \times \varepsilon$ matrix called the incidence matrix and the $\nu \times \nu$ matrix called the adjacency matrix of G .

Definition 15. The incidence matrix of G is the $\nu \times \varepsilon$ matrix $M(G) = [m_{ij}]$, where m_{ij} is the number of times (0, 1 or 2) that v_i and e_j are incident.

The incidence matrix of a graph is just a different way of specifying the graph. If $m_{ij} = 2$ then e_j is a loop incident with v_i in graph G .

Definition 16. The adjacency matrix of G is the $\nu \times \nu$ matrix $A(G) = [a_{ij}]$, where a_{ij} is the number of edges joining v_i and v_j .

A graph, its incidence matrix, and its adjacency matrix are shown in Fig.1.3.

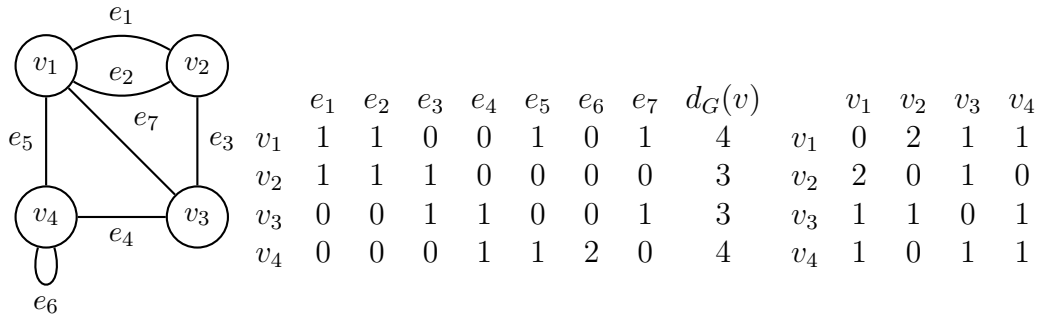


Figure 1.3: G , $M(G)$ and $A(G)$

Definition 17. A graph H is a subgraph of G (written $H \subseteq G$) if $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$, and ψ_H is the restriction of ψ_G to $E(H)$.

Definition 18. A graph H is a proper subgraph of G (written $H \subsetneq G$), if $H \subseteq G$ and $H \neq G$.

Suppose that V' is a nonempty subset of V . The subgraph of G whose vertex set is V' and whose edge set is the set of those edges of G that have both ends in V' is called the subgraph of G induced by V' and is denoted by $G[V']$; we say that $G[V']$ is an induced subgraph of G . The induced subgraph $G[V \setminus V']$ is denoted by $G - V'$; it is the subgraph obtained from G by deleting the vertices in V' with their incident edges. If $V' = \{v\}$ we write $G - v$ for $G - \{v\}$. Now suppose that E' is a nonempty subset of E . The subgraph of G whose vertex set is the set of ends of edges in E' and whose edge set is E' is called the subgraph of G induced by E' and is denoted by $G[E']$; $G[E']$ is an edge-induced subgraph of G . The spanning subgraph of G with edge set $E \setminus E'$ is written simply as $G - E'$; it is the subgraph obtained from G by deleting the edges in E' . Similarly, the graph obtained from G by adding a set of edges E' is denoted by $G + E'$. If $E' = \{e\}$ we write $G - e$ and $G + e$ instead of $G - \{e\}$ and $G + \{e\}$.

Let G_1 and G_2 be subgraphs of G . We say that G_1 and G_2 are disjoint if they have no vertex in common, and edge-disjoint if they have no edge in common. The union $G_1 \cup G_2$ of G_1 and G_2 is the subgraph with vertex set $V(G_1) \cup V(G_2)$ and edge set $E(G_1) \cup E(G_2)$; if G_1 and G_2 are disjoint, we sometimes denote their union by $G_1 + G_2$. The intersection $G_1 \cap G_2$ of G_1 and G_2 is defined similarly, but in this case G_1 and G_2 must have at least one vertex in common.

Definition 19. The degree $d_G(v)$ of a vertex v in G is the number of edges of G incident with v , each loop counting as two edges.

The degree of a vertex in G in connection with $M(G)$ can be written as: $d_G(v_i) = \sum_{1 \leq j \leq \varepsilon(G)} m_{ij}$ for $v_i \in V(G)$ where $1 \leq i \leq \nu(G)$. We denote by $\delta(G)$ and $\Delta(G)$ the minimum and maximum degrees, respectively, of vertices of G . Let us recall the following two results.

Theorem 1 ([10]). $\sum_{v \in V} d_G(v) = 2\varepsilon$

Corollary 1 ([10]). *In any graph, the number of vertices of odd degree is even.*

In Fig. 1.3, we can see that $\sum_{1 \leq i \leq 4} d(v_i) = 14$, while counting degrees of vertices, each edge contributes two degrees: one each to its end vertices. That is each edge is counted twice while counting the degrees of all vertices. Therefore the sum of degrees of all vertices in G is twice the number of edges in G , here the numbers of edges is 7.

Definition 20. *A graph G is k -regular if $d(v) = k$ for all $v \in V$; a regular graph is one that is k -regular for some k .*

Complete graphs K_n and complete bipartite graphs $K_{n,n}$ are regular, since we have K_n is $(n - 1)$ -regular and $K_{n,n}$ is n -regular.

Definition 21. *Let $k \in \mathbb{N}_0$. A walk (of length k) in a graph G is a finite non-empty alternating sequence $W = v_0 e_0 v_1 e_1 v_2 e_2 \dots v_{k-1} e_{k-1} v_k$ of vertices and edges in G such that $\psi_G(e_i) = v_i v_{i+1}$ for all $0 \leq i < k$. The length of the walk W is denoted by $\ell(W)$ Let us denote by $E(W)$ the set of edges which appear in W , and let us denote by $\lambda_W(e)$ the number of occurrences of $e \in E(G)$ in W . Set $V(W) = \{v_i \mid i = 0, \dots, k\}$.*

We say that W is a walk from v_0 to v_k , or a (v_0, v_k) -walk. The vertices v_0 and v_k are called the origin and terminus of W , respectively, and v_1, v_2, \dots, v_{k-1} its internal vertices. If $v_0 = v_k$ then walk is closed and open if they are different. We speak of a covering walk if $E(W) = E(G)$, whereas a $V(G)$ -covering walk only satisfies $V(W) = V(G)$. A covering walk is called a double-tracing if $\lambda_W(e) = 2$ for every $e \in E(G)$.

If $W = v_0 e_0 v_1 e_1 \dots v_{k-1} e_{k-1} v_k$ is a walk, then the walk $v_k e_{k-1} v_{k-1} \dots v_1 e_0 v_0$, obtained by reversing W , is denoted by W^R . If $W_1 = v_0 e_0 v_1 e_1 \dots v_{k-1} e_{k-1} v_k$, $W_2 = x_0 f_0 x_1 f_1 \dots x_{\ell-1} f_{\ell-1} x_\ell$ are walks and if $v_k = x_0$ then the concatenation of W_1 and W_2 at the vertex v_k is the walk $v_0 e_0 \dots v_{k-1} e_{k-1} v_k f_0 x_1 f_1 \dots x_{\ell-1} f_{\ell-1} x_\ell$, and it is denoted by $W_1 \cdot_{v_k} W_2$. Here we can note that $\ell(W_1) = k$, $\ell(W_2) = \ell$ and $\ell(W_1 \cdot_{v_k} W_2) = k + \ell$.

A subsequence of a walk W is a walk that can be derived from W , by deleting either its prefix or suffix or both, in the walk W . For example, $X = v_1e_5v_4e_5v_1$ is a subsequence of walk $W = v_0e_0v_1e_5v_4e_5v_1e_6v_4e_7v_2e_1v_1$ derived by deleting its prefix v_0e_0 and suffix $e_6v_4e_7v_2e_1v_1$. A section of a walk $W = v_0e_0v_1 \dots e_{k-1}v_k$ is a walk that is a subsequence $v_i e_i v_{i+1} e_{i+1} \dots v_{j-1} e_{j-1} v_j$ of W ; we refer to this subsequence as the (v_i, v_j) -section of W .

In a simple graph, a walk $v_0e_0 \dots e_{k-1}v_k$ is determined by the sequence $v_0v_1 \dots v_k$ of its vertices; hence a walk in a simple graph can be specified simply by its vertex sequence. Moreover, even in graphs that are not simple, we shall sometimes refer to a sequence of adjacent vertices as a ‘walk’. In such cases it is understood that the discussion is valid for every walk with that sequence.

Similar to $V(G)$, $E(G)$, $\nu(G)$ and $\varepsilon(G)$, the number of vertices of a walk W in G is called the *order* of the walk, denoted by $\nu(W)$ and the number of edges of the graph W is called the *size* of the walk, denoted by $\varepsilon(W)$.

Definition 22. Let $W = v_0e_0v_1e_1 \dots e_{k-1}v_k$ be a walk. W is called a trail if $e_i \neq e_j$ whenever $i \neq j$, $0 \leq i, j \leq k-1$; in this case $\ell(W) = \varepsilon(W)$.

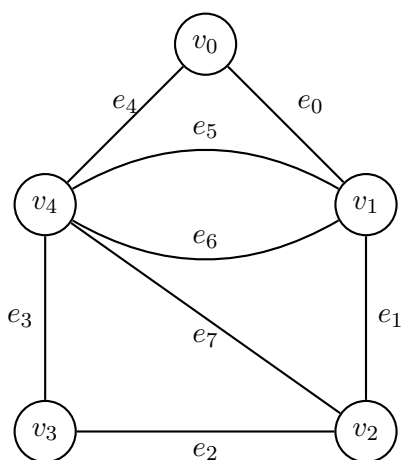
Remark 5. $W = v_0$ is the trivial trail and $\ell(W) = \varepsilon(W) = 0$. A second smallest trail can be a loop or an edge between two vertices. More explicitly, $W = v_0e_0v_1$ is a second smallest trail, if $v_0 = v_1$ then W is a loop and if $v_0 \neq v_1$ then W is a link between v_0 and v_1 . In both cases $\ell(W) = \varepsilon(W) = 1$.

Definition 23. An open trail satisfying $\ell(W) = |V(W)| - 1$ is a path.

In other words, an open trail in which no vertex appears (traversed) more than once is called a path. The number of edges, $\varepsilon(W)$, in a path W is called the length of the path. It immediately follows, then, that an edge which is not a loop is a path of length one. It should also be noted that a loop can be included in a trail but not in a path. The terminal vertices of a path are of degree one, and the rest of the vertices (intermediate vertices) are of degree two. This degree, of course, is counted only with respect to the edges included in the path and not the entire graph in which the path may be contained.

Fig.1.4 illustrates a walk, a trail and a path in a graph. We shall also use the word ‘path’ to denote a graph or subgraph whose vertices and edges are of a path. Just as with walks we sometimes use the term (v_0, v_k) -path to denote a path from v_0 to v_k .

Two vertices u and v of G are said to be *connected* if there is a (u, v) -path in G . Connectedness is an equivalence relation on the vertex set V . Thus there is a



Walk: $v_0e_0v_1e_5v_4e_5v_1e_6v_4e_7v_2e_1v_1$

Trail: $v_2e_2v_3e_3v_4e_7v_2e_1v_1e_6v_4$

Path: $v_3e_2v_2e_7v_4e_4v_0e_0v_1$

Figure 1.4: Walk, Trail and Path

partition of V into nonempty subsets $V_1, V_2, \dots, V_\omega$ such that two vertices u and v are connected if and only if both u and v belong to the same set V_i . The subgraphs $G[V_1], G[V_2], \dots, G[V_\omega]$ are called the *components* of G . If G has exactly one component, G is *connected*; otherwise G is *disconnected*. We denote the number of components of G by $\omega(G)$. Connected and disconnected graphs are depicted in Fig. 1.5 and Fig. 1.6.

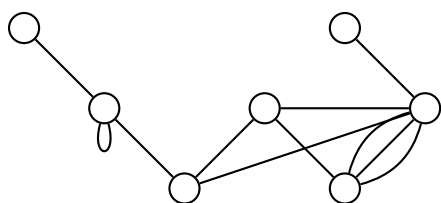


Figure 1.5: A connected graph

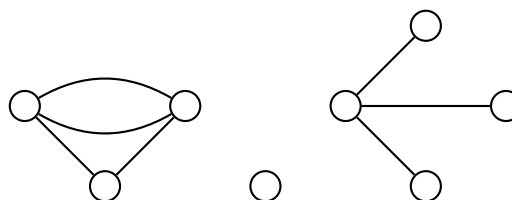


Figure 1.6: A disconnected graph with 3 components

Definition 24. Let $W = v_0e_0 \dots e_{k-1}v_k$ be a closed trail. A closed trail W is a cycle if $\varepsilon(W) = |V(W)|$; in this case we have $\ell(W) = k > 0$.²

Just as with paths we use the term ‘cycle’ to denote a graph that corresponds to a cycle.

Definition 25. A cycle of length k is called a k -cycle; a k -cycle is odd or even according as k is odd or even.

²Cycles are often called circuits, while closed trails are often called cycles. Note that a cycle must have at least one edge.

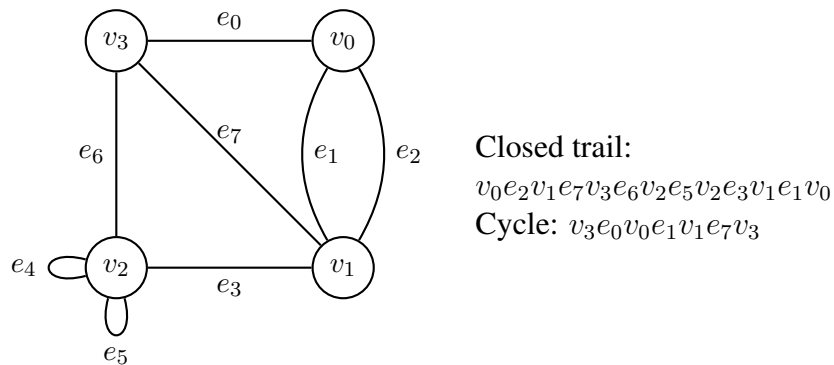


Figure 1.7: Closed trail and Cycle

A 3-cycle is often called a *triangle*. Examples of a closed trail and a cycle are given in Fig. 1.7.

Definition 26. A covering trail is called an Eulerian Trail.

Remark 6. Let G be the trivial graph then $W = v_0$ is the trivial Eulerian trail.

These trails named after Euler since he was the first to investigate the existence of such trails in graphs. In the earliest known paper on graph theory of Euler [24], he showed that it was impossible to cross each of the seven bridges of Königsberg once and only once during a walk through the town. Proving that such a walk is impossible amounts to showing that the graph of Fig. 1.8 contains no Eulerian trail.

Definition 27. A closed Eulerian trail is an Eulerian Tour.

Remark 7. $W = v_0 e_0 v_0$ is the smallest Eulerian tour.

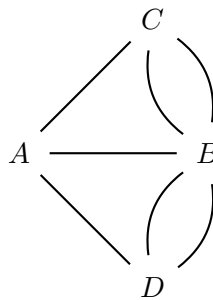


Figure 1.8: Graph Representation of Königsberg Bridge Problem

Definition 28. A graph is Eulerian if it contains an Eulerian tour.

Let us recall the following few results.

Theorem 2 ([10]). A nonempty connected graph is Eulerian if and only if it has no vertices of odd degree.

Corollary 2 ([10]). A connected graph has an Eulerian trail if and only if it has at most two vertices of odd degree.

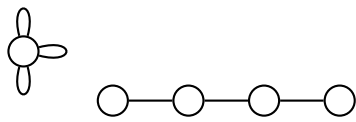


Figure 1.9: $n = 1$ and $n = 4$

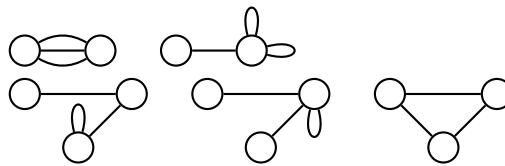


Figure 1.10: $n = 2$ and $n = 3$

Remark 8. A graph G with an Eulerian trail W with n vertices, where $4 \geq n \geq 1$ and $\varepsilon(G) = 3$ can be drawn in as many ways as in Figures 1.9 and 1.10.

Chapter 2

Eulerian Trails

In this chapter we try to answer a (first) question: How to place sets of Eulerian trails [24] in the Chomsky hierarchy? This was motivated by yet another (second) question: How to do the so called shuffle operation [43] that has been defined in 1970's and also its restrictions such as literal shuffle [8], balanced literal shuffle and shuffle operation on trajectories [81] to the Eulerian graphs? The second question has been answered in [91] using the pseudo-linear form (PLF) [102]. The first question has been answered by us in [28]. Here in this chapter we provide detailed proofs for some of the theorems in [28] and also few new results.

2.1 Formal Language Questions for Eulerian Trails

We consider connected graphs. If we draw all the vertices of a graph $G = (V, E, \psi)$ on a horizontal line, then we associate different integers to the vertices by a function called *pseudo-linear drawing*, which is defined as follows:

Definition 29. A pseudo-linear drawing (PLD) is an injective function $\phi : V \rightarrow \mathbb{Z}$.

Definition 30. $PLD_\phi(G)$ is a graph with vertex set $\phi(V)$ such that $PLD_\phi(G) \cong G$.

Definition 31. If ϕ is a PLD, then u is to the left of v in the drawing $PLD_\phi(G)$ if and only if $\phi(u) < \phi(v)$. Similarly u is to the right of v in the drawing $PLD_\phi(G)$ if and only if $\phi(v) < \phi(u)$.

Example 7. Let G be graph as in Fig. 2.1 and let ϕ be a PLD defined as $\phi(v_0) = 100$, $\phi(v_1) = 1000$ and $\phi(v_2) = 2000$ then $PLD_\phi(G)$ is as in Fig. 2.1.

Let us define the word representation of the Eulerian trails as follows:

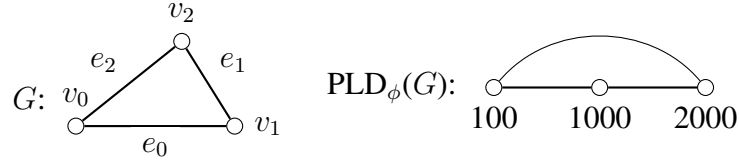


Figure 2.1: Graph G and $\text{PLD}_\phi(G)$ in Example 7

Definition 32. A connected graph $G = (V, E, \psi)$, with an Eulerian trail $W = v_0 e_0 v_1 e_1 \dots e_{k-1} v_k$ and any PLD $\phi : V \rightarrow \mathbb{Z}$, defines a word $w = \text{word}(G, \phi, W) \in \Sigma^*$, $\Sigma = \{\rightarrow, \leftarrow, |\}$ associated to G , ϕ and W as follows:

$$\text{word}(G, \phi, W) = \begin{cases} \varepsilon & \text{if } W = v_0 \\ w' w'', w'' = \rightarrow |^s & \text{if } s = \phi(v_k) - \phi(v_{k-1}) \geq 0 \wedge W \neq v_0 \\ w' w'', w'' = \leftarrow |^s & \text{if } s = \phi(v_{k-1}) - \phi(v_k) > 0 \wedge W \neq v_0 \end{cases}$$

Here $w' = \text{word}(G', \phi', W')$, where $W' = v_0 e_0 \dots e_{k-2} v_{k-1}$, $G' \subsetneq G$,

$$G' = \begin{cases} G - e_{k-1} & \text{if } d_G(v_k) > 1 \\ G - v_k & \text{if } d_G(v_k) = 1 \end{cases}$$

with $V(G') = V' = V$ if $G' = G - e_{k-1}$ and $V' = V - v_k$ if $G' = G - v_k$, $E(G') = E' \subsetneq E$, ψ' is the restriction of ψ to E' and ϕ' the restriction of ϕ to V' .

Note 1: In the above definition the trail W satisfies $W = W' \cdot_{v_{k-1}} W''$ where $\ell(W') = k - 1$, and $W'' = v_{k-1} e_{k-1} v_k$ where $\ell(W'') = 1$ to satisfy $\ell(W) = k$.

Note 2: $W'' = v_{k-1} e_{k-1} v_k$ is an Eulerian trail of the graph $G'' = (V'', E'', \psi'')$, where $V'' = \{v_k, v_{k-1}\}$, $E'' = \{e_{k-1}\}$, $\psi''(e_{k-1}) = v_{k-1} v_k$. Then G'' satisfies the condition that $G = G' \cup G''$ with $v_{k-1} \in V' \cap V''$. Here G' and G'' are edge-disjoint but not vertex-disjoint. Also G'' has a PLD ϕ'' , the restriction of ϕ to V'' , that is $\phi'' = \phi|_{V''}$. So we have $\text{word}(G'', \phi'', W'') = w''$. This gives the following observation.

Observation 1. $\text{word}(G, \phi, W) = \text{word}(G', \phi', W') \cdot \text{word}(G'', \phi'', W'')$.

Example 8. Let us consider a graph $G = (V, E, \psi)$, where $V = \{x, y\}$, $E = \{e\}$, $\psi(e) = xy$ with an Eulerian trail $W = xey$ and a PLD $\phi : V \rightarrow \mathbb{Z}$ which is given as $\phi(x) = z_0$ and $\phi(y) = z_1$, where $z_0, z_1 \in \mathbb{Z}$. By Definition 32 we find that

$$\text{word}(G, \phi, W) = \begin{cases} \rightarrow |^s & \text{if } z_0 + s = z_1, s > 0 \\ \leftarrow |^s & \text{if } z_1 + s = z_0, s > 0 \end{cases}$$

If $x = y$ then $z_0 = z_1$, then $s = 0$ and $\text{word}(G, \phi, W) = \rightarrow$.

Hence, G, ϕ , and an Eulerian trail W in G specify a word $word(G, \phi, W) = w$ over the alphabet $\Sigma = \{\rightarrow, \leftarrow, |\}$, called *Eulerian trace*. This gives the following formal definition of the *set of all Eulerian traces*.

Definition 33. $ET = \{w \in \{\rightarrow, \leftarrow, |\}^* \mid \exists G = (V, E, \psi) \text{ with an Eulerian trail } W = v_0 e_0 v_1 e_1 \dots e_{k-1} v_k \text{ and } \exists \text{ a PLD } \phi : V \rightarrow \mathbb{Z} \text{ such that } w = word(G, \phi, W)\}$.

Lemma 2. $ET \subseteq \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$, where $\Sigma = \{\rightarrow, \leftarrow, |\}$.

Proof. Let $w \in ET$. By Definition 33, there exists a graph $G = (V, E, \psi)$ with an Eulerian trail $W = v_0 e_0 v_1 e_1 \dots e_{k-1} v_k$ and there exists a PLD $\phi : V \rightarrow \mathbb{Z}$ such that $w = word(G, \phi, W)$. By Definition 32, we have two cases for w as follows: Case 1: If $w = \varepsilon$ then $w \in \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$. Case 2: If $w \neq \varepsilon$ then $w = w'w''$, where $w' = word(G', \phi', W')$ (here again $w' = \varepsilon$ or $w' \neq \varepsilon$ which we will see as sub cases of Case 2 below) and $w'' = \rightarrow |^s$, if $s = \phi(v_k) - \phi(v_{k-1}) \geq 0$ and $w'' = \leftarrow |^s$, if $s = \phi(v_{k-1}) - \phi(v_k) > 0$. This implies $w'' \in \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$. Again by Definition 32, we have two cases for w' as follows: Case 2a: If $w' = \varepsilon$ then $w = w''$. Case 2b: If $w' \neq \varepsilon$ then $w' = w'_1 w'_2$, where $w'_1 = word(G'_1, \phi'_1, W'_1)$ and $w'_2 = \rightarrow |^s$, if $s = \phi(v_k) - \phi(v_{k-1}) \geq 0$ and $w'_2 = \leftarrow |^s$, if $s = \phi(v_{k-1}) - \phi(v_k) > 0$. If $w'_1 = \varepsilon$ then $w' \in \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$. But if $w'_1 \neq \varepsilon$ then w'_1 will be decomposed to two words and we should continue to discuss again until we get a decomposition that has the first word as ε then the decomposition stops and we could see that the second word is in $\Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$. Now we conclude Case 2 that if $w \neq \varepsilon$ then $w \in \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$. Hence $ET \subseteq \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$. \square

Observation 2. Let us consider a word $w \in \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$, namely $w = \rightarrow |^s$. Let us reconsider the graph $G = (V, E, \psi)$ in Example 8. If $s > 0$ then we find $word(G, \phi, W) = \rightarrow |^s$ or $word(G, \phi, W) = \leftarrow |^s$. Similarly if $s = 0$ then $word(G, \phi, W) = \rightarrow$. So we observe that $w = \rightarrow |^{s_1}$, $s_1 \geq 0$ and $w = \leftarrow |^{s_2}$, $s_2 > 0$ corresponds to an Eulerian trail W with $\ell(W) = 1$ of a graph G with $\nu(G) = \nu(W) = n$, where $2 \geq n \geq 1$ and $\varepsilon(G) = 1$ that has a PLD ϕ such that $word(G, \phi, W) = w$.

Lemma 3. Let W_1 and W_2 be two Eulerian trails of graphs $G_1 = (V_1, E_1, \psi_1)$ and $G_2 = (V_2, E_2, \psi_2)$ respectively. Then $W_1 \cdot_{v_k} W_2$ is also an Eulerian trail of graph $G_1 \cup G_2$ with $v_k \in V_1 \cap V_2$ and v_k being the terminus of W_1 and origin of W_2 .

Proof. Let $W_1 = v_0 e_0 v_1 \dots e_{k-1} v_k$ and $W_2 = x_0 f_0 x_1 \dots f_{\ell-1} x_\ell$ be two Eulerian trails of graphs $G_1 = (V_1, E_1, \psi_1)$ and $G_2 = (V_2, E_2, \psi_2)$ respectively. We have $V_1 = \{v_0, \dots, v_k\}$ and $V_2 = \{x_0, \dots, x_\ell\}$. Assume $v_k = x_0$. Then $W_1 \cdot_{v_k} W_2 = v_0 e_0 \dots e_{k-1} v_k f_0 \dots f_{\ell-1} x_\ell$. Let $W = W_1 \cdot_{v_k} W_2$. Now W is an Eulerian trail of a

graph G , where $G = G_1 \cup G_2$ with $v_k \in V_1 \cap V_2$ and v_k being the terminus of W_1 and origin of W_2 . \square

Observation 3. Lemma 3 is true only for v_k being the terminus of W_1 and the origin of W_2 and it is not true for any $v_k \in V_1 \cap V_2$. For instance, let $W_1 = x_0 e_0 x_1 e_4 x_2 e_1 x_0 e_2 x_3$ and $W_2 = y_0 f_0 x_0 f_1 y_2$ (see Fig. 2.2). Here, x_0 is the vertex common in the vertex set of both W_1 and W_2 . But it is not both: terminus of W_1 and origin of W_2 , so $W_1 \cdot_{x_0} W_2$ is not an Eulerian trail (see Fig. 2.3).

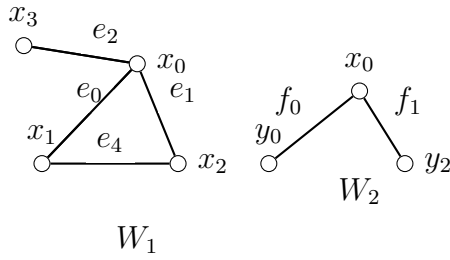


Figure 2.2: Eulerian trails W_1 and W_2

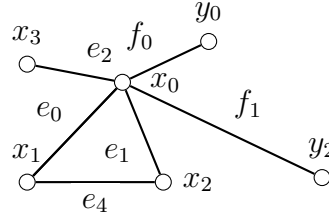


Figure 2.3: $W_1 \cdot_{x_0} W_2$

To prove the other inclusion of Lemma 2, let us define the homomorphism $h : \Sigma^* \rightarrow \{\rightarrow, \leftarrow\}^*$ such that $h(\rightarrow) = \rightarrow$, $h(\leftarrow) = \leftarrow$ and $h(|) = \varepsilon$. Let us consider $w \in \Sigma^* \setminus ((\{|)\} \Sigma^* \cup \{\leftarrow\})$. We have to prove that $w \in ET$. For this let us restate $\Sigma^* \setminus ((\{|)\} \Sigma^* \cup \{\leftarrow\}) \subseteq ET$ in more detail as in the following lemma.

Lemma 4. Let $w \in \Sigma^* \setminus ((\{|)\} \Sigma^* \cup \{\leftarrow\})$. Let $k \in \mathbb{N}_0$. If $|h(w)| = k$, then w corresponds to an Eulerian trail W with $\ell(W) = \varepsilon(W) = k$ of a graph $G = (V, E, \psi)$ with $\nu(G) = \nu(W) = n$ where $k + 1 \geq n \geq 1$ and $\varepsilon(G) = k$ that has a PLD $\phi : V \rightarrow \mathbb{Z}$ such that $\text{word}(G, \phi, W) = w$.

Proof. Let us prove this lemma by induction on k . Induction Basis: If $|h(w)| = k = 0$ then $w \in \{|)\}^*$ which implies $w = \varepsilon$. Let us consider an Eulerian trail W of the graph $G = (V, E, \psi)$ where $V = \{v_0\}$, $E = \emptyset$ and $\psi = \emptyset$ that has a PLD $\phi : V \rightarrow \mathbb{Z}$. We find that $\text{word}(G, \phi, W) = \varepsilon$. Moreover, w corresponds to the Eulerian trail W with $\ell(W) = \varepsilon(W) = 0$ of the graph G with $\nu(G) = \nu(W) = 1$ where G and W satisfies the conditions stated in Lemma 4.

Induction Hypothesis (IH): Assume that if $|h(w)| \leq k, k \geq 0$ then w corresponds to an Eulerian trail W with $\ell(W) = \varepsilon(W) = k$ of a graph $G = (V, E, \psi)$ with $\nu(G) = \nu(W) = n, k + 1 \geq n \geq 1$ and $\varepsilon(G) = k$ that has a PLD $\phi : V \rightarrow \mathbb{Z}$ such that $\text{word}(G, \phi, W) = w$.

Induction Step: We prove the lemma for $|h(w)| = k + 1$, $k \geq 0$. Let $w = w'w''$ where $|h(w')| = k$ and $|h(w'')| = 1$ and $w', w'' \in \Sigma^* \setminus ((\{\}\Sigma^*) \cup \{\leftarrow\})$.

$$\left. \begin{array}{l} \text{By IH } w' \text{ corresponds to an Eulerian trail } W' \text{ with } \ell(W') = \varepsilon(W') = k \\ \text{of a graph } G' = (V', E', \psi') \text{ with } \nu(G') = n', k + 1 \geq n' \geq 1 \wedge \varepsilon(G') = k \\ \text{that has a PLD } \phi' : V' \rightarrow \mathbb{Z} \text{ such that } \text{word}(G', \phi', W') = w'. \end{array} \right\} \quad (2.1)$$

By Observation 2, w'' corresponds to an Eulerian trail W'' with $\ell(W'') = \varepsilon(W'') = 1$ of a graph $G'' = (V'', E'', \psi'')$ with $\nu(G'') = n'', 2 \geq n'' \geq 1$ and $\varepsilon(G'') = 1$ that has a PLD $\phi'' : V'' \rightarrow \mathbb{Z}$ such that $\text{word}(G'', \phi'', W'') = w''$.

Let $W' = v_0e_0 \dots e_{k-1}v_k$. Let $W'' = xey$. For G'' we have two cases. Case 1: If $n'' = 1$ then $V'' = \{x\}$. Case 2: If $n'' = 2$ then $V'' = \{x, y\}$. $\phi''(x) = z_x$ and $\phi''(y) = z_y$, where $x, y \in V''$ and $z_x, z_y \in \mathbb{Z}$ and this implies that $x \neq y$ where as in Case 1 we have $x = y$.

We have $V' = \{v_0, \dots, v_k\}$. Let $\phi'(v_k) = \phi''(x)$, then $x = v_k$ (since ϕ' and ϕ'' are injective functions) and this implies $\phi'(v_k) = z_x$. Here $V' \cap V'' \neq \emptyset$. More precisely,

$$V' \cap V'' = \begin{cases} \{v_k\} & \text{if } z_y \notin \{\phi'(v_i), 0 \leq i \leq k\}, \\ \{v_k, y\} & \text{if } z_y \in \{\phi'(v_i), 0 \leq i \leq k\}. \end{cases}$$

From these two cases of $V' \cap V''$, we have two cases for $y \in V''$, respectively:

Case 1 : If $y \notin V'$ then $W' \cdot_{v_k} W'' = v_0e_0 \dots e_{k-1}v_k e y$ with $\ell(W' \cdot_{v_k} W'') = \ell(W') + \ell(W'') = k + 1$ of a graph $G = (V, E, \psi)$ where $V = V' \cup V''$, $E = E' \cup E''$, $\psi : E \rightarrow V^2$ is the incidence function such that $\psi|_{E'} = \psi'$ and $\psi|_{E''} = \psi''$, with

- $k + 2 \geq \nu(G) \geq 1$ since $\nu(G) = n' + (n'' - 1) = n' + 1 \leq (k + 1) + 1 = k + 2$ by IH from (2.1) and since $\nu(G) = n' + 1 \geq 1$,
- $\varepsilon(G) = k + 1$ since $E = E' \cup E'' \wedge E' \cap E'' = \emptyset$,

that has a PLD $\phi : (V' \cup V'') \rightarrow \mathbb{Z}$ where ϕ is obtained from both ϕ' and ϕ'' with $\phi'(v_k) = \phi''(x)$ (since $x = v_k \in V' \cap V''$) such that $\text{word}(G, \phi, W' \cdot_{v_k} W'') = \text{word}(G, \phi, W') \cdot \text{word}(G, \phi, W'') = w'w''$.

Case 2 : If $y \in V'$ then $W' \cdot_{v_k} W''$ with $\ell(W' \cdot_{v_k} W'') = \ell(W') + \ell(W'') = k + 1$ of a graph $G = (V, E, \psi)$ where $V = V' \cup V''$ and $E = E' \cup E''$ and $\psi : E \rightarrow V^2$ is the incidence function such that $\psi|_{E'} = \psi'$ and $\psi|_{E''} = \psi''$, with

- $k + 1 \geq \nu(G) \geq 1$ since $\nu(G) = n' \leq k + 1$ by IH from (2.1) and since $\nu(G) = n' \geq 1$,
- $\varepsilon(G) = k + 1$ since $E = E' \cup E'' \wedge E' \cap E'' = \emptyset$,

that has a PLD $\phi : (V' \cup V'') \rightarrow \mathbb{Z}$ where $\phi = \phi'$ such that $\text{word}(G, \phi, W' \cdot_{v_k} W'') = \text{word}(G, \phi, W') \cdot \text{word}(G, \phi, W'') = w'w''$.

By Lemma 3, as W' and W'' are Eulerian trails, $W' \cdot_{v_k} W''$ is also an Eulerian trail. As $v_k \in V' \cap V''$, we can also note that $G = G' \cup G''$ in both cases. Therefore, w with $|h(w)| = k + 1$ corresponds to an Eulerian trail W with $\ell(W) = \varepsilon(W) = k + 1$ of a graph $G = (V, E, \psi)$ with $\nu(G) = \nu(W) = n$, $k + 2 \geq n \geq 1$ and $\varepsilon(G) = k + 1$ that has a PLD $\phi : V \rightarrow \mathbb{Z}$ such that $\text{word}(G, \phi, W) = w$. Hence, by the principle of induction the lemma is proven. \square

Theorem 3. $ET = \Sigma^* \setminus ((\{\}\Sigma^*) \cup \{\leftarrow\})$.

Proof. The proof follows from Lemmas 2 and 4. \square

Illustration 1. Let us now illustrate Lemma 4. Let $w = \rightarrow | \rightarrow | \leftarrow ||$. Then $w \in \Sigma^* \setminus (\{\}\Sigma^*)$. As $|h(w)| = 3$, by Lemma 4, w corresponds to an Eulerian trail W with $\ell(W) = \varepsilon(W) = 3$ of a graph $G = (V, E, \psi)$ with $\nu(G) = \nu(W) = n$ where $4 \geq n \geq 1$ and $\varepsilon(G) = 3$ that has a PLD $\phi : V \rightarrow \mathbb{Z}$ such that $\text{word}(G, \phi, W) = \rightarrow | \rightarrow | \leftarrow ||$.

Now let us see how the word $\rightarrow | \rightarrow | \leftarrow ||$ corresponds to a graph G , that describes a triangle (G is a 3-cycle): Let us split the word $w = \rightarrow | \rightarrow | \leftarrow ||$ such that $w' = \rightarrow | \rightarrow |$ and $w'' = \leftarrow ||$. Then $|h(w')| = 2$ and $|h(w'')| = 1$.

Now again split $w' = \rightarrow | \rightarrow |$ such that $w'_1 = \rightarrow |$, $w'_2 = \rightarrow |$. Then $|h(w'_1)| = 1$ and $|h(w'_2)| = 1$. By Lemma 4, w'_1 corresponds to an Eulerian trail W'_1 with $\ell(W'_1) = 1$ of a graph $G'_1 = (V'_1, E'_1, \psi'_1)$ with $\nu(G'_1) = \nu(W'_1) = n'_1$ where $2 \geq n'_1 \geq 1$ and $\varepsilon(G'_1) = 1$ that has a PLD $\phi'_1 : V'_1 \rightarrow \mathbb{Z}$ such that $\text{word}(G'_1, \phi'_1, W'_1) = w'_1$. Now let us find the graph G'_1 such that $\text{word}(G'_1, \phi'_1, W'_1) = w'_1 = \rightarrow |$.

Let $W'_1 = v_0 e_0 v_1$ be the Eulerian trail of the graph G'_1 with $V'_1 = \{v_0, v_1\}$, $E'_1 = \{e_0\}$ and $\psi'_1(e_0) = v_0 v_1$. Please note that up to this moment we cannot conclude whether $n'_1 = 1$ ($v_0 = v_1$) or $n'_1 = 2$ ($v_0 \neq v_1$).

Let $\phi'_1(v_0) = z_0$ and $\phi'_1(v_1) = z_1$, where $z_0, z_1 \in \mathbb{Z}$. Let us write $w'_1 = \rightarrow |^{s_1}$, where $s_1 = 1$. Since $s_1 = 1$ by Definition 32 we have $s_1 = z_1 - z_0 = 1 \geq 0$, which implies that $z_0 \neq z_1$, and that implies $v_0 \neq v_1$ ($n'_1 = 2$).

Therefore we can draw the graph G'_1 as an edge between two vertices v_0 and v_1 (see Fig. 2.4). Also since $z_0 < z_1$, v_0 is to the left of v_1 in PLD $\phi'_1(G'_1)$ (see Fig. 2.5). Also note that $z_0 + s_1 = z_1$.

Similarly, By Lemma 4, w'_2 corresponds to an Eulerian trail W'_2 with $\ell(W'_2) = 1$ of a graph $G'_2 = (V'_2, E'_2, \psi'_2)$ with $\nu(G'_2) = \nu(W'_2) = n'_2$ where $2 \geq n'_2 \geq 1$ and $\varepsilon(G'_2) = 1$ that has a PLD $\phi'_2 : V'_2 \rightarrow \mathbb{Z}$ such that $\text{word}(G'_2, \phi'_2, W'_2) = w'_2$. Now let us find the graph G'_2 such that $\text{word}(G'_2, \phi'_2, W'_2) = w'_2 \Rightarrow \rightarrow |$.

Let $W'_2 = v_1 e_1 v_2$ be the Eulerian trail of the graph G'_2 with $V'_2 = \{v_1, v_2\}$, $E'_2 = \{e_1\}$ and $\psi'_2(e_1) = v_1 v_2$. Let $\phi'_2(v_1) = z_1$ and $\phi'_2(v_2) = z_2$, where $z_1, z_2 \in \mathbb{Z}$. Let us write $w'_2 \Rightarrow \rightarrow |^{s_2}$, where $s_2 = 1$. Since $s_2 = 1$ we have by Definition 32 that $s_2 = z_2 - z_1 = 1 \geq 0$, which implies that $z_1 \neq z_2$, and that implies $v_1 \neq v_2$ ($n'_2 = 2$). Therefore we can draw the graph G'_2 as an edge between two vertices v_1 and v_2 (see Fig. 2.4). Also since $z_1 < z_2$, v_1 is to the left of v_2 in $\text{PLD}_{\phi'_2}(G'_2)$ (see Fig. 2.5). Also note that $z_1 + s_2 = z_2$.

We have $w' \Rightarrow \rightarrow | \rightarrow |$. Then $|h(w')| = 2$. By Lemma 4 w' corresponds to an Eulerian trail W' with $\ell(W') = \varepsilon(W') = 2$ of a graph $G' = (V', E', \psi')$ with $\nu(G') = \nu(W') = n'$ where $2 + 1 \geq n' \geq 1$ and $\varepsilon(G') = 2$ that has a PLD $\phi' : V' \rightarrow \mathbb{Z}$ such that $\text{word}(G', \phi', W') = w'$.

Let us concatenate the Eulerian trails $W'_1 = v_0 e_0 v_1$ and $W'_2 = v_1 e_1 v_2$ and name it as W' , so $W'_1 \cdot_{v_1} W'_2 = W'$. Then we have $W' = v_0 e_0 v_1 e_1 v_2$ and $\ell(W') = 2$. We know from W'_1 and W'_2 that $v_0 \neq v_1$ and $v_1 \neq v_2$. Also we know that $z_0 + s_1 = z_1$, $s_1 = 1$ and $z_1 + s_2 = z_2$, $s_2 = 1$. This implies that $z_0 \neq z_2$. Since $z_0 \neq z_2$, $v_0 \neq v_2$. Therefore we can draw G' as in Fig. 2.4.

By Lemma 4, w'' corresponds to an Eulerian trail W'' with $\ell(W'') = 1$ of a graph $G'' = (V'', E'', \psi'')$ with $\nu(G'') = \nu(W'') = n''$ where $2 \geq n'' \geq 1$ and $\varepsilon(G'') = 1$ that has a PLD $\phi'' : V'' \rightarrow \mathbb{Z}$ such that $\text{word}(G'', \phi'', W'') = \leftarrow ||$. Now let us find the graph G'' such that $\text{word}(G'', \phi'', W'') = w'' = \leftarrow ||$.

Let $W'' = v_2 e_2 v_3$ be the Eulerian trail of the Graph G'' with $V'' = \{v_2, v_3\}$, $E'' = \{e_2\}$ and $\psi''(e_2) = v_2 v_3$. Let $\phi''(v_2) = z_2$ and $\phi''(v_3) = z_3$, where $z_2, z_3 \in \mathbb{Z}$. Let us write $w'' = \leftarrow |^{s_3}$, where $s_3 = 2$. Since $s_3 = 2$ we have by Definition 32 that $s_3 = z_2 - z_3 = 2 \geq 0$, which implies that $z_2 \neq z_3$, and that implies $v_2 \neq v_3$ ($n'' = 2$). Therefore we can draw the graph G'' as an edge between two vertices v_2 and v_3 (see Fig. 2.4). Also since $z_3 < z_2$, v_3 is to the left of v_2 in the $\text{PLD}_{\phi''}(G'')$ (see Fig. 2.5). Also note that $z_3 + s_3 = z_2$.

Finally, we have that $w \Rightarrow \rightarrow | \rightarrow | \leftarrow ||$. Then $|h(w)| = 3$. By Lemma 4 w corresponds to an Eulerian trail W with $\ell(W) = \varepsilon(W) = 3$ of a graph $G = (V, E, \psi)$ with $\nu(G) = \nu(W) = n$ where $3 + 1 \geq n \geq 1$ and $\varepsilon(G) = 3$ that has a PLD $\phi : V \rightarrow \mathbb{Z}$ such that $\text{word}(G, \phi, W) = w$.

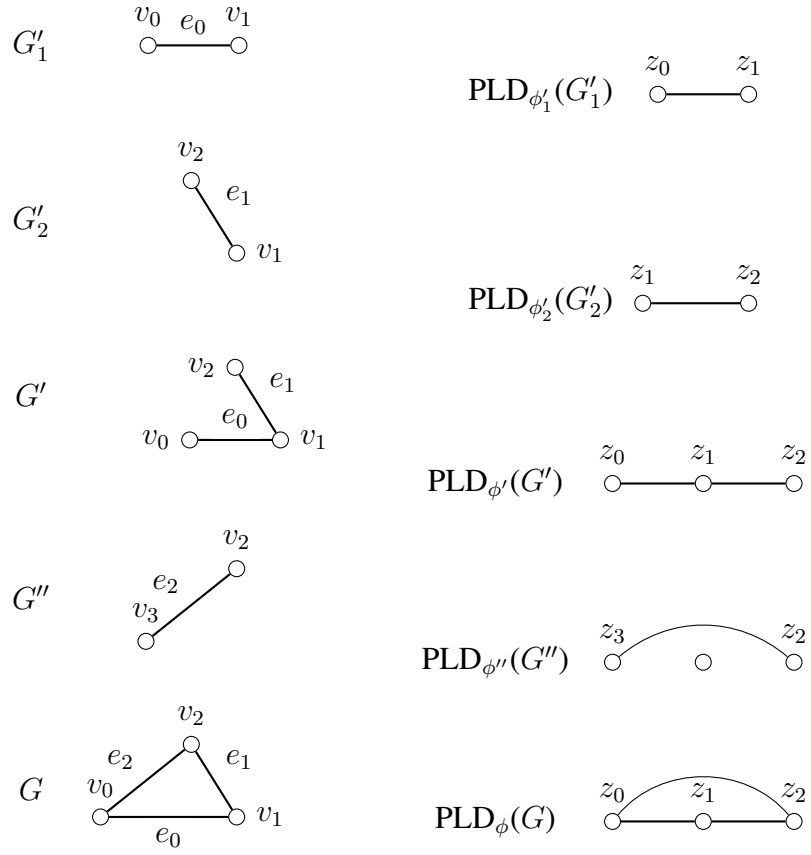


Figure 2.4: Graphs G, G', G'', G'_1, G'_2

Figure 2.5: PLDs of Graphs in Fig.2.4

Let us concatenate the Eulerian trails $W' = v_0e_0v_1e_1v_2$, and $W'' = v_2e_2v_3$ and name it as W , so $W' \cdot_{v_2} W'' = W$. Then we have $W = v_0e_0v_1e_1v_2e_2v_3$ and $\ell(W) = 3$. We know from W' and W'' that $v_0 \neq v_1, v_1 \neq v_2, v_0 \neq v_2$ and $v_2 \neq v_3$. Also we know that $z_0 + s_1 = z_1, s_1 = 1$ and $z_1 + s_2 = z_2, s_2 = 1$ and $z_2 - s_3 = z_3, s_3 = 2$. This implies that $z_0 = z_3$ and $z_1 \neq z_3$. Since $z_0 = z_3, v_0 = v_3$ and since $z_1 \neq z_3, v_1 \neq v_3$. Therefore we can draw G as in Fig. 2.4. Also since $z_0 < z_1 < z_2, v_0$ is to the left of v_1 and v_1 is to the left of v_2 in the $\text{PLD}(G)$ (see Fig. 2.5). Also note that $z_0 + 1 = z_1, z_1 + 1 = z_2$ and $z_2 - 2 = z_0$.

2.2 Standard PLD

In Section 2.1 we saw that given any connected graph $G = (V, E, \psi)$ with an Eulerian trail W and any PLD $\phi : V \rightarrow \mathbb{Z}$, we find a word $w \in ET$. Conversely, given any word $w \in ET$ we find the connected graph $G = (V, E, \psi)$ with an Eulerian trail W that has a PLD $\phi : V \rightarrow \mathbb{Z}$ such that $word(G, \phi, W) = w$.

Given a word $w \in ET$ we find the graph G that satisfies $word(G, \phi, W) = w$, but there can be infinitely many graphs for $w \in ET$ which are isomorphic to G . Let a graph, say G_w be a representative of the equivalence class of isomorphic graphs of G , where $G_w = (V, E, \psi)$ with $V = \{v_i\}$, $0 \leq i \leq k$, $E = \{e_j\}$, $0 \leq j \leq k - 1$ and $\psi(e) = uv$, $e \in E$ and $u, v \in V$. By the illustration of Lemma 4, we have seen that the PLD ϕ plays a vital role in identifying the right graph $G = (V, E, \psi)$, for the given word $w \in ET$.

Now, it is natural to think about the uniqueness in PLD, by having a standard variant of it, and to have a unique graph G_w that has an Eulerian trail W with the standard PLD ϕ_w that gives a unique $PLD_{\phi_w}(G_w)$. Notice that, as $\phi(v_0) \in \mathbb{Z}$ and it (v_0 is the start vertex (origin) of W) is not specified with $w \in ET$, so there are infinitely many PLDs ϕ such that $w = word(G, \phi, W)$, but all are obtained from another by shifting along the horizontal line and hence describe the same graph. Let us now see an example for this and then we will formally define how this shifting gives us the same graph.

Example 9. Let $w = \rightarrow | \rightarrow |$ then by Definition 33 there exists a graph $G = (V, E, \psi)$ where $V = \{v_0, v_1, v_2\}$, $E = \{e_0, e_1\}$, $\psi(e_0) = v_0v_1$ and $\psi(e_1) = v_1v_2$ with an Eulerian trail $W = v_0e_0v_1e_1v_2$ and there exists a PLD $\phi : V \rightarrow \mathbb{Z}$ such that $w = word(G, \phi, W)$. For the graph G let us define two PLDs $\phi_1 : V \rightarrow \mathbb{Z}$ and $\phi_2 : V \rightarrow \mathbb{Z}$. Let us define the first PLD ϕ_1 as follows: $\phi_1(v_0) = 0$ $\phi_1(v_1) = 1$ and $\phi_1(v_2) = 2$. Similarly one can define another PLD say ϕ_2 as follows: $\phi_2(v_0) = 100$ $\phi_2(v_1) = 101$ and $\phi_2(v_2) = 102$.

Like in example 9 there are infinitely many PLDs that can be defined for $w = \rightarrow | \rightarrow |$. But this does not affect the properties of G and hence of the Eulerian trail W . This makes us to think on putting our $PLD_{\phi}(G)$ in infinitely many ways in the horizontal line (number line) with respect to the infinitely many PLDs. In Fig. 2.6, we have only provided $PLD_{\phi_1}(G)$ and $PLD_{\phi_2}(G)$ for graph G with respect to the PLDs ϕ_1 and ϕ_2 in example 9, but one can think of placing $PLD_{\phi}(G)$ in number line anywhere between $-\infty$ and ∞ with respect to ϕ that has been defined.

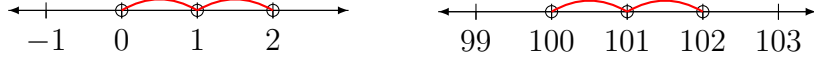


Figure 2.6: $\text{PLD}(G)$ with respect to PLDs ϕ_1 and ϕ_2 in Example 9

In the end it is going to be the same graph G where ever we wish to place it in the number line. That is we are horizontally translating the $\text{PLD}_\phi(G)$ in the number line. Let us formally define this horizontal translation of $\text{PLD}_\phi(G)$ in terms of PLD as follows:

Definition 34. Let $\phi : V \rightarrow \mathbb{Z}$ be a PLD of the graph $G = (V, E, \psi)$ with an Eulerian trail W with $\ell(W) = \varepsilon(G) = k$ and $\nu(W) = \nu(G) = n$ where $k + 1 \geq n \geq 1$, $k \in \mathbb{N}_0$. Let $\phi(v_i) = z_i$ where $v_i \in V$ and $0 \leq i \leq k$. The translation of ϕ , $\phi_T : V \rightarrow \mathbb{Z}$ is defined by $\phi_T(v_i) = z_i + C$ where $C \in \mathbb{Z}$.

Note: In Definition 34 if $C > 0$ then the translation happened towards right whereas if $C < 0$ then the translation happened towards left and if $C = 0$ then $\phi_T = \phi$ that is in this case no translation has happened.

Definition 35. Let \mathcal{P} be the set of all PLDs. Let $\phi_1, \phi_2 \in \mathcal{P}$. We say that ϕ_1, ϕ_2 are equivalent, $\phi_1 \simeq \phi_2$ if and only if $\phi_1(v) - \phi_2(v) = d$ for some $d \in \mathbb{Z}$ and for all $v \in V$.

As the name chosen in the previous definition suggests, we can prove that the relation \simeq on \mathcal{P} is an equivalence relation:

Lemma 5. $\simeq \subseteq \mathcal{P} \times \mathcal{P}$ is an equivalence relation.

Proof. We have to check the three properties of an equivalence relation.

Reflexivity: Let $\phi \in \mathcal{P}$. If $d = 0$ then $\phi(v) - \phi(v) = 0$, $v \in V$. So, $\phi \simeq \phi$.

Symmetry: Let $\phi_1, \phi_2 \in \mathcal{P}$. Assume that $\phi_1 \simeq \phi_2$. Then $\phi_1(v) - \phi_2(v) = d_1$ for some $d_1 \in \mathbb{Z}$ and for all $v \in V$. Then $\phi_2(v) - \phi_1(v) = -(\phi_1(v) - \phi_2(v)) = -d_1$ and $-d_1 \in \mathbb{Z}$ and for all $v \in V$. Thus, $\phi_2 \simeq \phi_1$.

Transitivity: Let $\phi_1, \phi_2, \phi_3 \in \mathcal{P}$. Assume that $\phi_1 \simeq \phi_2$ and $\phi_2 \simeq \phi_3$. Then $\phi_1(v) - \phi_2(v) = d_1$ for some $d_1 \in \mathbb{Z}$ and for all $v \in V$. Also, $\phi_2(v) - \phi_3(v) = d_2$ for some $d_2 \in \mathbb{Z}$ and for all $v \in V$. Then $\phi_1(v) - \phi_3(v) = (\phi_1(v) - \phi_2(v)) + (\phi_2(v) - \phi_3(v)) = d_1 + d_2$. Let $d_3 = d_1 + d_2$. Since $d_1, d_2 \in \mathbb{Z}$, $d_3 \in \mathbb{Z}$. Therefore, $\phi_1 \simeq \phi_3$. \square

Since \simeq on the set \mathcal{P} is an equivalence relation. For each $\phi_1 \in \mathcal{P}$ we can define the equivalence class of ϕ_1 , denoted by $[\phi_1]$, to be the set

$$[\phi_1] = \{\phi_2 \in \mathcal{P} \mid \phi_2 \simeq \phi_1\}.$$

Let us now define the standard PLD ϕ_w as follows:

Definition 36. A PLD ϕ_w of G_w is said to be a standard PLD, if and only if the start vertex v_0 of the Eulerian trail W of G_w satisfies the condition that $\phi_w(v_0) = 0$.

Here after, for every $w \in ET$ we refer to one unique graph $G_w = (V_w, E_w, \psi_w)$ where $V_w = \{v_0, \dots, v_k\}$, $E_w = \{e_0, \dots, e_{k-1}\}$, and $\psi_w(e) = uv$, where $e \in E_w$ and $u, v \in V_w$. This unique graph G_w is isomorphic to all the graphs G that satisfy $word(G, \phi, W) = w$ and the PLD ϕ can be translated to the standard PLD ϕ_w by Definition 34. This uniqueness is important for us to have one single graph G_w and one single PLD ϕ_w that leads to have one PLD(G_w) for the given word $w \in ET$. For example, for the graph G in example 9 we can find the unique graph G_w in this example $G_w = G$ and ϕ_w is obtained by translation of ϕ_2 (see Fig. 2.6) to ϕ_1 by Definition 34 and in this example $\phi_1 = \phi_w$.

2.3 Eulerian Traces

In this section we will be discussing about certain subsets of ET that describe few properties of the Eulerian traces.

Lemma 6. $ET \in \mathcal{REG}$.

Proof. By Theorem 3, $ET = \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$. Since ET is expressed by a regular expression $\Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$, $\Sigma = \{\rightarrow, \leftarrow, |\}$, $ET \in \mathcal{REG}$. \square

We define the set $ET^\circ \subseteq ET$ of descriptions of Eulerian graphs as follows:

Definition 37. $ET^\circ = \{w \in \Sigma^* : w \in ET \wedge G_w \text{ is Eulerian}\}$, $\Sigma = \{\rightarrow, \leftarrow, |\}$.

Let us define a deterministic blind one-counter machine M that accepts ET° , $L(M) = ET^\circ$. Let $M = (Q, \Sigma, \delta, q, F, 1)$ where

- $Q = \{q, q', r, l\}$ is the finite set of states,
- $\Sigma = \{\rightarrow, \leftarrow, |\}$ is the input alphabet,
- $\delta : Q \times \Sigma \times \{-1, 0, 1\} \rightarrow Q \times \{-1, 0, 1\}$ as described in the state diagram (see Fig. 2.7),
- $q \in Q$ is the initial state,
- $F = \{r, l\} \subseteq Q$ is the set of final states.

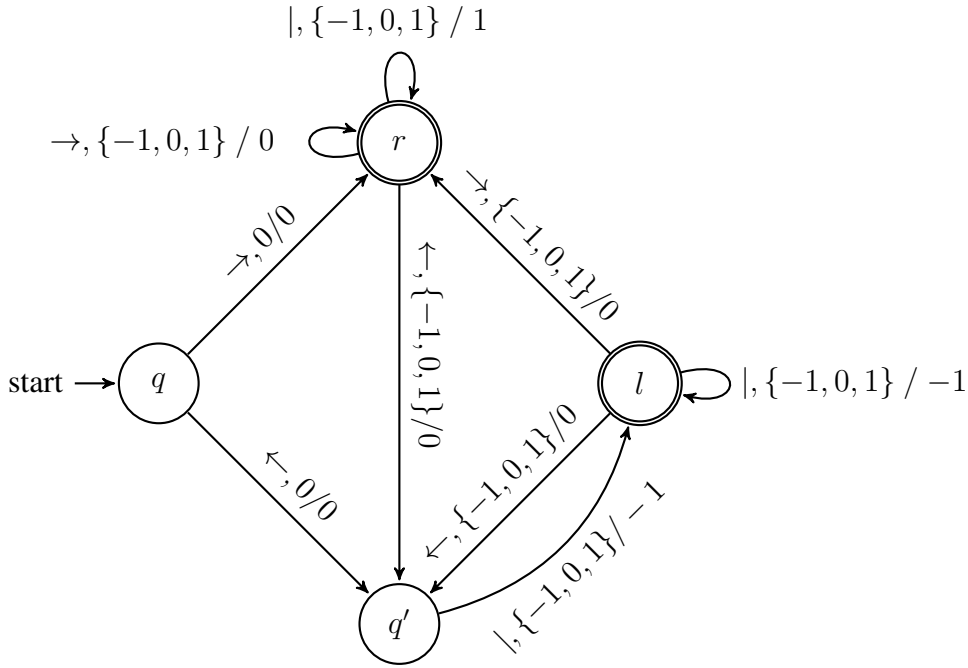


Figure 2.7: Deterministic Blind One-Counter Machine M that accepts ET°

The language accepted by M is $L(M) = \{w \in \Sigma^* : w \in ET \setminus \{\varepsilon\} \wedge c_0(w) \vdash_M^+ c_f(w)\}$, i. e., $L(M) = \{w \in \Sigma^* : w \in ET \setminus \{\varepsilon\} \wedge \exists m > 0 : (q, w, 0) \vdash_M^m (f, \varepsilon, 0)\}$, $f \in F$. Let us prove in Lemmas 9 and 10 that $L(M) = ET^\circ$ for the deterministic blind one-counter machine M in Fig. 2.7.

Lemma 7. $G = (V, E, \psi)$, $E = \{e_0, \dots, e_{k-1}\}$ is Eulerian if and only if $G' = (V', E', \psi')$, $V' \subseteq V$, $E' = \{e_0, \dots, e_{k-3}\} \cup \{e'\}$, $e' \notin E$ is Eulerian, where G' is obtained from G by replacing two adjacent edges from G by a single edge that is not in G .

Proof. G' is obtained from G by selecting two adjacent edges, say, $e_1, e_2 \in E$ and by replacing them by $e' \in E'$ such that $\psi_G(e_1) = uv$, $\psi_G(e_2) = vw$ and $\psi_{G'}(e') = uw$. Then $\psi|_{E \setminus \{e_1, e_2\}} = \psi'|_{E' \setminus \{e'\}}$.

$$V' = \begin{cases} V & \text{if } d_G(v) > 2, \\ V \setminus \{v\} & \text{if } d_G(v) = 2. \end{cases}$$

By Theorem 2, G is Eulerian if and only if $d_G(x)$ is even, for all $x \in V$. This implies that $d_G(x) = d_{G'}(x)$, for all $x \in V \setminus \{v\}$. And $d_{G'}(v) = d_G(v) - 2$ is even if and only if $d_G(v)$ is even. G is Eulerian implies that $d_G(x)$ is even for

all $x \in V$. Due to the construction of G' , we have $d_{G'}(x)$ is even for all $x \in V'$ which implies G' is Eulerian. G' is Eulerian implies that $d_{G'}(x)$ is even for all $x \in V'$. This implies that $d_G(x) = d_{G'}(x)$ for all $x \in V$. This implies that G is Eulerian. \square

Lemma 8. *Let M be the machine in Fig. 2.7. Let $w \in ET$ with a decomposition $w = w'w''$ where $w', w'' \in ET$ and for some $\ell_1 \geq 0, \ell_2 \geq 1$*

if $w'' = \rightarrow |\ell_1$, then $(q, w, 0) \vdash_M^+(r, \varepsilon, 0) \iff G_w$ is Eulerian,

and if $w'' = \leftarrow |\ell_2$, then $(q, w, 0) \vdash_M^+(l, \varepsilon, 0) \iff G_w$ is Eulerian.

Proof. Let M be the machine in Fig. 2.7. Let $w \in ET$ with a decomposition $w = w'w''$ where $w', w'' \in ET$ and for some $\ell_1 \geq 0, \ell_2 \geq 1$, if $w'' = \rightarrow |\ell_1$, then our claim is $(q, w, 0) \vdash_M^+(r, \varepsilon, 0) \iff G_w$ is Eulerian, and if $w'' = \leftarrow |\ell_2$, then our claim is $(q, w, 0) \vdash_M^+(l, \varepsilon, 0) \iff G_w$ is Eulerian. We prove both claims by induction on $|w|$. Note that for the case that $|w| = 1$, since ET does not have a word \leftarrow , for the induction basis we have the following:

Induction Basis: If $|w| = 1$, then there is a decomposition $w = w'w''$ where $w', w'' \in ET$ and for some $\ell_1 \geq 0$,

if $w'' = \rightarrow |\ell_1$, then

$$\begin{aligned} (q, w, 0) \vdash_M^+(r, \varepsilon, 0) &\iff w' = \varepsilon \wedge w'' = \rightarrow |^0 \text{ (since } |w| = 1) \\ &\iff w = \rightarrow \text{ (since } w = w'w'') \\ &\iff G_w \text{ is Eulerian (see Remark 7)} \end{aligned}$$

Induction Hypothesis: Assume that the lemma is true for $|w| \leq n, n > 0$.

Induction Step: For $|w| = n + 1, n > 0$ with a decomposition $w = w'w''$ where $w', w'' \in ET$ and for some $\ell_1 \geq 0, \ell_2 \geq 1$, if $w'' = \rightarrow |\ell_1$, then $(q, w, 0) \vdash_M^+(r, \varepsilon, 0) \iff G_w$ is Eulerian and if $w'' = \leftarrow |\ell_2$, then $(q, w, 0) \vdash_M^+(l, \varepsilon, 0) \iff G_w$ is Eulerian.

For $w \in ET$ with the decomposition $w = w'w'', w', w'' \in ET$ and for some $\ell_1 \geq 0, \ell_2 \geq 1$, we have the following two cases (Right Ending and Left Ending):

Case 1 (Right Ending): if $w'' = \rightarrow |\ell_1$, then based on w' we have the following cases:

Case 1a: If $w' = \varepsilon$ then $w = w''$ and if $\ell_1 = 0$ then $(q, w, 0) \vdash_M^{\ell_1+1}(r, \varepsilon, \ell_1) \iff G_w$ is Eulerian (by Induction Basis), but if $\ell_1 > 0$ then $(q, w, 0) \vdash_M^{\ell_1+1}(r, \varepsilon, \ell_1)$ implies that $(r, \varepsilon, \ell_1) \neq (r, \varepsilon, 0)$ which implies $w \notin L(M)$ and this implies G_w is

not Eulerian (whereas it has an Eulerian trail). See Fig. 2.8.

Case 1b: If $w' \neq \varepsilon$, then there is a decomposition $w' = w'_1 w'_2$, where $w'_1 \in ET$ and $w'_2 = \leftarrow |^{p_2}$ or $w'_2 = \rightarrow |^{p_1}$, $p_1 \geq 0$, $p_2 \geq 1$. Let $|w'_1| = p'_1$, $p'_1 \geq 0$.

Let $u = w'_2 w''$. Then, we have $w = w'_1 u$ and $|u| = p_1 + \ell_1 + 2$ or $|u| = p_2 + \ell_1 + 2$ also $|w| = p'_1 + p_1 + \ell_1 + 2$ or $|w| = p'_1 + p_2 + \ell_1 + 2$. Let $p_1 + \ell_1 + 2 = s_1$ and $p_2 + \ell_1 + 2 = s_2$ then $|u| = s_1$ or $|u| = s_2$ and $|w| = p'_1 + |u|$. Also since $|w| = n + 1$, $n > 0$ we have $n = p'_1 + |u| - 1$.

Now, let us define

$$\bar{u} = \begin{cases} \rightarrow |^{p_1 + \ell_1} & \text{if } u = \rightarrow |^{p_1} \rightarrow |^{\ell_1} \\ \rightarrow |^{\ell_1 - p_2} & \text{if } u = \leftarrow |^{p_2} \rightarrow |^{\ell_1} \wedge p_2 < \ell_1 \\ \leftarrow |^{p_2 - \ell_1} & \text{if } u = \leftarrow |^{p_2} \rightarrow |^{\ell_1} \wedge \ell_1 < p_2 \end{cases}$$

Claim I :

$$\forall t \in \{q, r, l\} \forall i, j \in \mathbb{Z} ([\exists f \in \{r, l\} : (t, u, i) \vdash_M^{|u|} (f, \varepsilon, j)] \iff [\exists \bar{f} \in \{r, l\} : (t, \bar{u}, i) \vdash_M^{|\bar{u}|} (\bar{f}, \varepsilon, j)])]$$

Proof of Claim I : Let $t \in \{q, r, l\}$ and $i, j \in \mathbb{Z}$. We prove by induction on $|u|$.

Induction Basis: If $|u| = 2$ then $u = \rightarrow \rightarrow$. By definition \bar{u} , the \bar{u} corresponding to u is nothing but $\bar{u} = \rightarrow$. So we have the following:

If $u = \rightarrow \rightarrow$ and if $\bar{u} = \rightarrow$ then

$$\begin{aligned} & \exists f \in \{r, l\} : (t, u, i) \vdash_M^2 (f, \varepsilon, j) \\ \iff & \exists f \in \{r, l\} : (t, \rightarrow \rightarrow, i) \vdash_M^2 (f, \varepsilon, j) \text{ (since } u = \rightarrow \rightarrow) \\ \iff & (t, \rightarrow \rightarrow, i) \vdash_M^1 (r, \rightarrow, i) \vdash_M^1 (f, \varepsilon, j) \wedge f = r \wedge i = j \text{ (by } \delta \text{ of } M) \\ \iff & (t, \rightarrow \rightarrow, i) \vdash_M^2 (f, \varepsilon, j) \wedge f = r \wedge i = j \text{ (by definition of } \vdash_M^2) \\ \iff & (t, \rightarrow, i) \vdash_M^1 (\bar{f}, \varepsilon, j) \wedge \bar{f} = r \wedge i = j \text{ (by } \delta \text{ transitions of } M) \\ \iff & (t, \bar{u}, i) \vdash_M^1 (\bar{f}, \varepsilon, j) \wedge \bar{f} = r \wedge i = j \text{ (since } \bar{u} = \rightarrow) \end{aligned}$$

Induction Hypothesis: We assume that Claim I is true for all $|u| \leq s_1 - 1$, $s_1 > 2$ and $|u| \leq s_2 - 1$, $s_2 > 3$.

Induction Step: We now prove that Claim I is true for all $|u| = s_1$ and $|u| = s_2$. Depending upon u we have three cases.

Case 1 : Let us consider the case if $u = \rightarrow |^{p_1} \rightarrow |^{\ell_1}$ then $\bar{u} = \rightarrow |^{p_1 + \ell_1}$. Let $u = u_1 |^{\ell_1}$, where $u_1 = \rightarrow |^{p_1} \rightarrow$ then $\bar{u} = \bar{u}_1 |^{\ell_1}$, where $\bar{u}_1 = \rightarrow |^{p_1}$.

$$\begin{aligned}
& \exists f \in \{r, l\} : (t, u, i) \vdash_M^{|u|} (f, \varepsilon, j) \\
\iff & \exists f \in \{r, l\} : (t, \rightarrow |^{p_1} \rightarrow |^{\ell_1}, i) \vdash_M^{p_1+\ell_1+2} (f, \varepsilon, j) \text{ (since } u = \rightarrow |^{p_1} \rightarrow |^{\ell_1}) \\
\iff & \exists f \in \{r, l\} : (t, u_1|^{\ell_1}, i) \vdash_M^{p_1+\ell_1+2} (f, \varepsilon, j) \text{ (as } u = u_1|^{\ell_1}, u_1 = \rightarrow |^{p_1} \rightarrow) \\
\iff & (t, u_1|^{\ell_1}, i) \vdash_M^{p_1+2} (r, |^{\ell_1}, i + p_1) \vdash_M^{\ell_1} (f, \varepsilon, j) \wedge f = r \wedge j = i + p_1 + \ell_1 \\
& \text{(by } \delta \text{ transitions of } M) \\
\iff & (t, \bar{u}_1|^{\ell_1}, i) \vdash_M^{p_1+1} (r, |^{\ell_1}, i + p_1) \vdash_M^{\ell_1} (\bar{f}, \varepsilon, j) \wedge \bar{f} = r \wedge j = i + p_1 + \ell_1 \\
& \text{(by applying IH to } u_1 \text{ and by } \delta \text{ transitions of } M) \\
\iff & \exists \bar{f} \in \{r, l\} : (t, \bar{u}_1|^{\ell_1}, i) \vdash_M^{p_1+\ell_1+1} (\bar{f}, \varepsilon, j) \\
& \text{(by definition of reflexive transitive closure of } \vdash_M^+) \\
\iff & \exists \bar{f} \in \{r, l\} : (t, \rightarrow |^{p_1+\ell_1}, i) \vdash_M^{p_1+\ell_1+1} (\bar{f}, \varepsilon, j) \text{ (as } \bar{u} = \bar{u}_1|^{\ell_1}, \bar{u}_1 = \rightarrow |^{p_1}) \\
\iff & \exists \bar{f} \in \{r, l\} : (t, \bar{u}, i) \vdash_M^{|u|} (\bar{f}, \varepsilon, j) \text{ (since } \bar{u} = \rightarrow |^{p_1+\ell_1})
\end{aligned}$$

Case 2 : If $u = \leftarrow |^{p_2} \rightarrow |^{\ell_1} \wedge p_2 < \ell_1$ then $\bar{u} = \rightarrow |^{\ell_1-p_2}$. Let $u = u_1|^{\ell_1}$, where $u_1 = \leftarrow |^{p_2} \rightarrow$ then $\bar{u} = \bar{u}_1|^{\ell_1}$, where $\bar{u}_1 = \rightarrow |^{-p_2}$.

$$\begin{aligned}
& \exists f \in \{r, l\} : (t, u, i) \vdash_M^{|u|} (f, \varepsilon, j) \\
\iff & \exists f \in \{r, l\} : (t, \leftarrow |^{p_2} \rightarrow |^{\ell_1}, i) \vdash_M^{p_2+\ell_1+2} (f, \varepsilon, j) \\
& \text{(since } u = \leftarrow |^{p_2} \rightarrow |^{\ell_1} \wedge p_2 < \ell_1) \\
\iff & \exists f \in \{r, l\} : (t, u_1|^{\ell_1}, i) \vdash_M^{p_2+\ell_1+2} (f, \varepsilon, j) \text{ (as } u = u_1|^{\ell_1}, u_1 = \leftarrow |^{p_2} \rightarrow) \\
\iff & (t, u_1|^{\ell_1}, i) \vdash_M^{p_2+2} (r, |^{\ell_1}, i + p_2) \vdash_M^{\ell_1} (f, \varepsilon, j) \wedge f = r \wedge j = i + p_2 + \ell_1 \\
& \text{(by } \delta \text{ transitions of } M) \\
\iff & (t, \bar{u}_1|^{\ell_1}, i) \vdash_M^{-p_2+1} (r, |^{\ell_1}, i - p_2) \vdash_M^{\ell_1} (\bar{f}, \varepsilon, j) \wedge \bar{f} = r \wedge j = i - p_2 + \ell_1 \\
& \text{(by applying IH to } u_1 \text{ and by } \delta \text{ transitions of } M) \\
\iff & \exists \bar{f} \in \{r, l\} : (t, \bar{u}_1|^{\ell_1}, i) \vdash_M^{-p_2+\ell_1+1} (\bar{f}, \varepsilon, j) \\
& \text{(by definition of reflexive transitive closure of } \vdash_M^+) \\
\iff & \exists \bar{f} \in \{r, l\} : (t, \rightarrow |^{\ell_1-p_2}, i) \vdash_M^{\ell_1-p_2+1} (\bar{f}, \varepsilon, j) \\
& \text{(since } \bar{u} = \bar{u}_1|^{\ell_1}, \bar{u}_1 = \rightarrow |^{-p_2}) \\
\iff & \exists \bar{f} \in \{r, l\} : (t, \bar{u}, i) \vdash_M^{|u|} (\bar{f}, \varepsilon, j) \text{ (since } \bar{u} = \rightarrow |^{\ell_1-p_2})
\end{aligned}$$

Case 3 : If $u = \leftarrow |^{p_2} \rightarrow |^{\ell_1} \wedge \ell_1 < p_2$ then $\bar{u} = \leftarrow |^{p_2-\ell_1}$. Let $u = u_1|^{\ell_1}$, where $u_1 = \leftarrow |^{p_2} \rightarrow$ then $\bar{u} = \bar{u}_1|^{-\ell_1}$, where $\bar{u}_1 = \leftarrow |^{p_2}$.

$$\begin{aligned}
& \exists f \in \{r, l\} : (t, u, i) \vdash_M^{|u|} (f, \varepsilon, j) \\
\iff & \exists f \in \{r, l\} : (t, \leftarrow^{p_2} \rightarrow^{|\ell_1|}, i) \vdash_M^{p_2 + \ell_1 + 2} (f, \varepsilon, j) \\
& \text{(since } u = \leftarrow^{p_2} \rightarrow^{|\ell_1|} \wedge \ell_1 \leq p_2) \\
\iff & \exists f \in \{r, l\} : (t, u_1^{|\ell_1|}, i) \vdash_M^{p_2 + \ell_1 + 2} (f, \varepsilon, j) \text{ (as } u = u_1^{|\ell_1|}, u_1 = \leftarrow^{p_2} \rightarrow) \\
\iff & (t, u_1^{|\ell_1|}, i) \vdash_M^{p_2 + 2} (r, |\ell_1|, i + p_2) \vdash_M^{\ell_1} (f, \varepsilon, j) \wedge f = r \wedge j = i + p_2 + \ell_1 \\
& \text{(by } \delta \text{ transitions of } M) \\
\iff & (t, \bar{u}_1^{|\ell_1|}, i) \vdash_M^{p_2 + 1} (l, |\ell_1|, i + p_2) \vdash_M^{\ell_1} (\bar{f}, \varepsilon, j) \wedge \bar{f} = l \wedge j = i + p_2 - \ell_1 \\
& \text{(by applying IH to } u_1 \text{ and by } \delta \text{ transitions of } M) \\
\iff & \exists \bar{f} \in \{r, l\} : (t, \bar{u}_1^{|\ell_1|}, i) \vdash_M^{p_2 - \ell_1 + 1} (\bar{f}, \varepsilon, j) \\
& \text{(by definition of reflexive transitive closure of } \vdash_M^+) \\
\iff & \exists \bar{f} \in \{r, l\} : (t, \leftarrow^{p_2 - \ell_1}, i) \vdash_M^{p_2 - \ell_1 + 1} (\bar{f}, \varepsilon, j) \\
& \text{(since } \bar{u} = \bar{u}_1^{-|\ell_1|}, \bar{u}_1 = \leftarrow^{p_2}) \\
\iff & \exists \bar{f} \in \{r, l\} : (t, \bar{u}, i) \vdash_M^{|u|} (\bar{f}, \varepsilon, j) \text{ (since } \bar{u} = \leftarrow^{p_2 - \ell_1})
\end{aligned}$$

So, from all the three cases we can conclude that Claim I is true for all $|u| = s_1$ and $|u| = s_2$. Before we start Claim II recall that Claim I and II are different from each other as $w = w'_1 u$, we can note that Claim I is on u where as Claim II is on w .

Claim II : $\exists f \in \{r, l\} : (q, w, 0) \vdash_M^{|w|} (f, \varepsilon, 0) \iff \exists \bar{f} \in \{r, l\} : (q, \bar{w}, 0) \vdash_M^{|\bar{w}|} (\bar{f}, \varepsilon, 0)$.

Proof of Claim II : Let $\bar{w} = w'_1 \bar{u}$. We prove the claim by induction on $|w|$. We know that $|w| = p'_1 + |u|$ and $|\bar{w}| = p'_1 + |\bar{u}|$. Also we have $|w| = n + 1, n > 0$. Let $n + 1 = m$.

Induction Basis: If $|w| = 2$ then $w = \rightarrow^0 \rightarrow^0$. i. e., $w = \rightarrow \rightarrow$. Since $|w| = 2$ and $w = w'_1 u$, we have $w'_1 = \varepsilon$ and $w = u$ which implies that the Induction Basis follows as same as the Induction Basis in Claim I with $i = j = 0$ and $t = q$.

Induction Hypothesis: Assume that Claim II is true for $|w| \leq m, m > 1$.

Induction Step: We have to prove that Claim II is true for $|w| = m + 1, m > 1$. Since $w = w'_1 u$, if $w'_1 = \varepsilon$ then $p'_1 = 0$ which implies $w = u$ and this implies that $|w| = |u|$ also $|\bar{w}| = |\bar{u}|$, so in this case the Induction Step also follows as same as the Induction Step in Claim I.

But if $w'_1 \neq \varepsilon$ then $p'_1 > 0$ which implies that $|w| = p'_1 + |u|$ and $|\bar{w}| = p'_1 + |\bar{u}|$. Now depending upon u and the corresponding \bar{u} we have the following three cases.

Case 1 : Let us consider the case if $u \Rightarrow |^{p_1} \rightarrow |^{\ell_1}$ then $\bar{u} \Rightarrow |^{p_1 + \ell_1}$.

Let $u = u_1|^{\ell_1}$, where $u_1 \Rightarrow |^{p_1} \rightarrow$ then $\bar{u} = \bar{u}_1|^{\ell_1}$, where $\bar{u}_1 \Rightarrow |^{p_1}$.

$$\begin{aligned}
& \exists f \in \{r, l\} : (q, w, 0) \vdash_M^{|w|} (f, \varepsilon, 0) \\
\iff & \exists f \in \{r, l\} : (q, w'_1 u, 0) \vdash_M^{p'_1 + |u|} (f, \varepsilon, 0) \text{ (since } w = w'_1 u \text{ and } p'_1 > 0) \\
\iff & \exists f \in \{r, l\} : (q, w'_1 \rightarrow |^{p_1} \rightarrow |^{\ell_1}, 0) \vdash_M^{p'_1 + s} (f, \varepsilon, 0) \\
& \text{(since } u \Rightarrow |^{p_1} \rightarrow |^{\ell_1} \wedge |u| = s) \\
\iff & \exists f \in \{r, l\} : (q, w'_1 u_1|^{\ell_1}, 0) \vdash_M^{p'_1 + s} (f, \varepsilon, 0) \text{ (as } u = u_1|^{\ell_1}, u_1 \Rightarrow |^{p_1} \rightarrow) \\
\iff & \exists f' \in \{r, l\} : (q, w'_1 u_1|^{\ell_1}, 0) \vdash_M^{p'_1} (f', u_1|^{\ell_1}, i) \vdash_M^s (f, \varepsilon, 0) \\
& \wedge f = r \wedge i = -(p_1 + \ell_1) \text{ (by } \delta \text{ transitions of } M) \\
\iff & \exists f' \in \{r, l\} : (q, w'_1 \bar{u}_1|^{\ell_1}, 0) \vdash_M^{p'_1} (f', \bar{u}_1|^{\ell_1}, i) \\
& \vdash_M^{p_1 + 1} (r, |^{\ell_1}, i + p_1) \vdash_M^{\ell_1} (\bar{f}, \varepsilon, 0) \\
& \wedge \bar{f} = r \wedge j = -(p_1 + \ell_1) + (p_1 + \ell_1) = 0 \\
& \text{(by applying IH to } w'_1 u_1 \text{ and by } \delta \text{ transitions of } M) \\
\iff & \exists f' \in \{r, l\} \exists \bar{f} \in \{r, l\} : (q, w'_1 \bar{u}_1|^{\ell_1}, 0) \vdash_M^{p'_1 + p_1 + \ell_1 + 1} (f, \varepsilon, 0) \\
& \text{(by definition of reflexive transitive closure of } \vdash_M^+) \\
\iff & \exists \bar{f} \in \{r, l\} : (q, w'_1 \rightarrow |^{p_1 + \ell_1}, 0) \vdash_M^{p'_1 + p_1 + \ell_1 + 1} (\bar{f}, \varepsilon, 0) \text{ (as } \bar{u}_1 \Rightarrow |^{p_1}) \\
\iff & \exists \bar{f} \in \{r, l\} : (q, w'_1 \bar{u}, 0) \vdash_M^{p'_1 + |\bar{u}|} (\bar{f}, \varepsilon, 0) \\
& \text{(since } \bar{u} \Rightarrow |^{p_1 + \ell_1} \wedge |\bar{u}| = p_1 + \ell_1 + 1) \\
\iff & \exists \bar{f} \in \{r, l\} : (q, \bar{w}, 0) \vdash_M^{|\bar{w}|} (\bar{f}, \varepsilon, 0) \text{ (since } \bar{w} = w'_1 \bar{u} \text{ and } p'_1 > 0)
\end{aligned}$$

Similarly we can have the remaining two cases:

Case 2 : If $u \Leftarrow |^{p_2} \rightarrow |^{\ell_1} \wedge p_2 < \ell_1$ then $\bar{u} \Rightarrow |^{\ell_1 - p_2}$.

Case 3 : If $u \Leftarrow |^{p_2} \rightarrow |^{\ell_1} \wedge \ell_1 < p_2$ then $\bar{u} \Leftarrow |^{p_2 - \ell_1}$.

Now from these three cases we can conclude that Claim II is true for all $|w| = m + 1, m > 1$.

Case 2 (Left Ending): if $w'' = \leftarrow |^{\ell_2}$, then based on $w' = \varepsilon$ and $w' \neq \varepsilon$ we have the following cases:

Case 2a: If $w' = \varepsilon$ then $w = w''$ and if $\ell_2 = 1$ then $(q, w, 0) \vdash_M^{\ell_2+1} (l, \varepsilon, \ell_2)$ implies that $(l, \varepsilon, \ell_2) \neq (l, \varepsilon, 0)$ which implies $w \notin L(M)$ and this implies G_w is not Eulerian (whereas it has an Eulerian trail). See Fig. 2.8.

Case 2b: Now, let us define \bar{u} for left ending as we have defined in Case 1b for right ending.

$$\bar{u} = \begin{cases} \leftarrow |^{p_2+\ell_2} & \text{if } u = \leftarrow |^{p_2} \leftarrow |^{\ell_2} \\ \leftarrow |^{\ell_2-p_1} & \text{if } u = \rightarrow |^{p_1} \leftarrow |^{\ell_2} \wedge p_1 < \ell_2 \\ \rightarrow |^{p_1-\ell_2} & \text{if } u = \rightarrow |^{p_1} \leftarrow |^{\ell_2} \wedge \ell_2 \leq p_1 \end{cases}$$

Case 2b will also have Claim I and II similar to Case 1b, where u and \bar{u} will have \leftarrow and \rightarrow instead of \rightarrow and \leftarrow respectively also p_1 is replaced by p_2 and ℓ_1 is replaced by ℓ_2 (compare the definitions of \bar{u} in Case 1b and Case 2b). Having these changes in Claim I and II of Case 1b, according counter values and state information follows, so we do not prove formally the Claims I and II for Case 2b. Now after having these two cases (Right and Left Ending) we continue to do the induction step of our lemma as follows:

We have $|w| = m, m > 1$ and $|w| = p'_1 + |u|, |u| > 1$. Also we have $|\bar{w}| = p'_1 + |\bar{u}|$. Let $|\bar{w}| = b, b > 0$. We can apply the Induction Hypothesis for \bar{w} as follows:

For $|\bar{w}| = b, b > 0$ with a decomposition $\bar{w} = w'_1 \bar{u}$ where $w'_1, \bar{u} \in ET$ and for some $\ell_1 \geq 0, \ell_2 \geq 1$, if $\bar{u} = \rightarrow |^{\ell_1}$, then $(q, \bar{w}, 0) \vdash_M^+ (r, \varepsilon, 0) \iff G_{\bar{w}}$ is Eulerian and if $\bar{u} = \leftarrow |^{\ell_2}$, then $(q, \bar{w}, 0) \vdash_M^+ (l, \varepsilon, 0) \iff G_{\bar{w}}$ is Eulerian.

From Claim II of Case 1b, we have $(q, w, 0) \vdash_M^{|w|} (r, \varepsilon, 0) \iff (q, \bar{w}, 0) \vdash_M^{|\bar{w}|} (r, \varepsilon, 0)$ and this implies that $(q, w, 0) \vdash_M^+ (r, \varepsilon, 0) \iff (q, \bar{w}, 0) \vdash_M^+ (r, \varepsilon, 0)$ since $|\bar{w}| = b, b > 0$ and $|w| = m, m > 1$.

And from Claim II of Case 2b we have $(q, w, 0) \vdash_M^{|w|} (l, \varepsilon, 0) \iff (q, \bar{w}, 0) \vdash_M^{|\bar{w}|} (l, \varepsilon, 0)$ and this implies that $(q, w, 0) \vdash_M^+ (l, \varepsilon, 0) \iff (q, \bar{w}, 0) \vdash_M^+ (l, \varepsilon, 0)$ since $|\bar{w}| = b, b > 0$ and $|w| = m, m > 1$.

So, for $|\bar{w}| = b, b > 0$ with a decomposition $\bar{w} = w'_1 \bar{u}$ where $w'_1, \bar{u} \in ET$ and for $|w| = m, m > 1$ with a decomposition $w = w' w''$ where $w', w'' \in ET$ and for some $\ell \geq 0$,

if $w'' \Rightarrow |\ell_1$, then

$$\begin{aligned} (q, w, 0) \vdash_M^+ (r, \varepsilon, 0) &\iff (q, \bar{w}, 0) \vdash_M^+ (r, \varepsilon, 0) \text{ (by Claim II of Case 1b)} \\ &\iff G_{\bar{w}} \text{ is Eulerian (by IH)} \\ &\iff G_w \text{ is Eulerian (by Lemma 7)} \end{aligned}$$

if $w'' \Leftarrow |\ell_2$, then

$$\begin{aligned} (q, w, 0) \vdash_M^+ (l, \varepsilon, 0) &\iff (q, \bar{w}, 0) \vdash_M^+ (l, \varepsilon, 0) \text{ (by Claim II of Case 2b)} \\ &\iff G_{\bar{w}} \text{ is Eulerian (by IH)} \\ &\iff G_w \text{ is Eulerian (by Lemma 7)} \end{aligned}$$

So, we can conclude that for $|w| = n + 1$, $n > 0$ with the decomposition $w = w'w''$ where $w', w'' \in ET$ and for some $\ell_1 \geq 0$, $\ell_2 \geq 1$, if $w'' \Rightarrow |\ell_1$, then $(q, w, 0) \vdash_M^+ (r, \varepsilon, 0) \iff G_w$ is Eulerian, and if $w'' \Leftarrow |\ell_2$, then $(q, w, 0) \vdash_M^+ (l, \varepsilon, 0) \iff G_w$ is Eulerian.

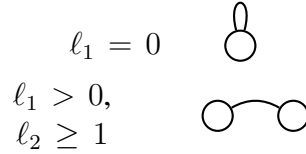


Figure 2.8: Case 1a and Case 2a

□

Lemma 9. Let M be the machine in Fig. 2.7. Let $w \in ET \setminus \{\varepsilon\}$. Then, $(q, w, 0) \vdash_M^+ (f, \varepsilon, 0), f \in F \iff G_w$ is Eulerian.

Proof. Consider the machine M in Fig. 2.7. Let $w \in ET \setminus \{\varepsilon\}$. Any accepting computation of M will start in state q and will make one transition to state r or state l and never return to state q . Thus it is sufficient to find the conditions on w such that w is accepted by M by reaching final states r, l . Those conditions are for all $w \in ET \setminus \{\varepsilon\}$ $(q, w, 0) \vdash_M^+ (r, \varepsilon, 0)$ or $(q, w, 0) \vdash_M^+ (l, \varepsilon, 0)$. By Lemma 8 we proved the following statements:

If $(q, w, 0) \vdash_M^+ (r, \varepsilon, 0)$ then G_w is Eulerian.

If $(q, w, 0) \vdash_M^+ (l, \varepsilon, 0)$ then G_w is Eulerian.

If G_w is Eulerian then $(q, w, 0) \vdash_M^+ (r, \varepsilon, 0)$ or $(q, w, 0) \vdash_M^+ (l, \varepsilon, 0)$.

Hence we have $(q, w, 0) \vdash_M^+ (f, \varepsilon, 0), f \in F \iff G_w$ is Eulerian.

□

Lemma 10. *Let M be the machine in Fig. 2.7. Then, $L(M) = ET^\circ$*

Proof. The language accepted by M ,

$$\begin{aligned} L(M) &= \{w \in \Sigma^* : w \in ET \setminus \{\varepsilon\} \wedge (q, w, 0) \vdash_M^+ (f, \varepsilon, 0)\} \text{ (By Definition 2)} \\ &= \{w \in \Sigma^* : w \in ET \wedge G_w \text{ is Eulerian}\} \text{ (By Lemma 9)} \\ &= ET^\circ \text{ (By Definition 37)}. \end{aligned}$$

□

Theorem 4. *ET° is a deterministic blind one-counter language.*

Proof. By Lemma 10, there exists a deterministic blind one-counter machine that accepts ET° hence ET° is a deterministic blind one-counter language. □

We now define formally, $ET_{\text{loop-free}}$ by both semantic and syntactic descriptions.

Definition 38. $ET_{\text{loop-free}} = \{w \in \Sigma^* : w \in ET \setminus \{\varepsilon\} \wedge G_w \text{ has no loops}\}.$

The above semantic description of $ET_{\text{loop-free}}$ can be given syntactically as follows:
 $\{w \in \Sigma^* : w \in ET \setminus \{\varepsilon\} \wedge w \in ((\{\rightarrow\} \cup \{\leftarrow\})\{\|\}^+)^+\}.$

Let us prove in Lemma 11 that semantic and syntactic descriptions of $ET_{\text{loop-free}}$ are equivalent. If $E = ((\rightarrow + \leftarrow)|^+)^+$ then $L(E) = ((\{\rightarrow\} \cup \{\leftarrow\})\{\|\}^+)^+$ and as mentioned in Remark 1 we use E for $L(E)$.

Lemma 11. $\forall w \in ET \setminus \{\varepsilon\} [w \in ((\rightarrow + \leftarrow)|^+)^+ \iff G_w \text{ has no loops}].$

Proof. Let $w \in ET \setminus \{\varepsilon\}$. We first prove that $w \in ((\rightarrow + \leftarrow)|^+)^+ \implies G_w$ has no loops. We prove this by induction on $|h(w)|$, where $h : \Sigma^* \rightarrow \{\rightarrow, \leftarrow\}^*$ such that $h(\rightarrow) = \rightarrow$, $h(\leftarrow) = \leftarrow$ and $h(|) = \varepsilon$. Before we start the induction, note that the smallest $w \in ((\rightarrow + \leftarrow)|^+)^+$ are nothing but $w \in (\rightarrow + \leftarrow)|$. So we start our basis with $|h(w)| = 1$.

Induction Basis: If $|h(w)| = 1$ then $w \in \rightarrow|^+$ or $w \in \leftarrow|^+$. If $w \in \rightarrow|^s$, $s \geq 1$ then G_w is the single edge that connects the start vertex v_0 to vertex v_1 satisfying $\phi_w(v_0) = 0$ and $\phi_w(v_1) = s$ with v_1 becoming the end vertex, i. e., describing a right move by s steps in the $\text{PLD}(G_w)$. Similarly, If $w \in \leftarrow|^s$, $s \geq 1$ then G_w is the single edge that connects the start vertex v_0 to the vertex v_1 satisfying $\phi_w(v_0) = 0$ and $\phi_w(v_1) = -s$, with v_1 becoming the end vertex, i. e., describing a left move by s steps in the $\text{PLD}(G_w)$. In both cases G_w is the single edge between the two vertices v_0 and v_1 , that has no loops, because $|\phi_w(v_0) - \phi_w(v_1)| = s \geq 1$.

Induction Hypothesis: Assume that for all $w \in ((\rightarrow + \leftarrow)|^+)^+$ with $|h(w)| \leq n$, $n \geq 1$, we know that G_w has no loops.

Induction Step: Consider $w \in ((\rightarrow + \leftarrow)|^+)^+$ with $|h(w)| = n + 1$, $n \geq 1$. Our claim is to show that G_w has no loops. Let $w = xa$ with $x \in ((\rightarrow + \leftarrow)|^+)^+$, $|h(x)| = n$ and a needs to start with \rightarrow or \leftarrow . Since $|h(x)| = n$ and $|h(w)| = n + 1$, $n \geq 1$ we have $|h(xa)| = |h(x)| + |h(a)| = n + 1$ that implies $|h(a)| = 1$. This implies that $a \in (\rightarrow + \leftarrow)|^+$.

By induction hypothesis G_x has no loops. Let $X = v_0e_0 \dots e_{k-1}v_k$ be a Eulerian trail of G_x with the standard PLD ϕ_x . Now we have two cases for a as follows:

Case 1: If $a \in \rightarrow|^+$, then the Eulerian trail $X \cdot_{v_k} v_k e_k v_{k+1}$ of G_{xa} , where $e_k \notin \{e_0, \dots, e_{k-1}\}$ has no loops as both Eulerian trails X and $v_k e_k v_{k+1}$ does not have any loops. **Case 2:** If $a \in \leftarrow|^+$, then as in Case 1, G_{xa} has no loops.

Therefore, $w \in ((\rightarrow + \leftarrow)|^+)^+$ with $|h(w)| = n + 1$, $n \geq 1$ implies G_w has no loops.

Now, secondly we prove that G_w has no loops implies $w \in ((\rightarrow + \leftarrow)|^+)^+$.

Let us consider the graph G_w with an Eulerian trail W that has the standard PLD ϕ_w that has no loops. By Definition 32, we have $\text{word}(G_w, \phi_w, W) = w$. Now our aim is to prove that $w \in ((\rightarrow + \leftarrow)|^+)^+$. We prove this by induction on $\ell(W)$. Before we start the induction, note that $\ell(W) = 0$ is excluded since we have considered $w \in ET \setminus \{\varepsilon\}$.

Induction Basis: If $\ell(W) = 1$ then $W = v_0e_0v_1$ and $\phi_w(v_0) = 0$ since W of G_w has no loops $\phi_w(v_1) \neq 0$ which implies that $\phi_w(v_1) = s$, $s \in \mathbb{Z} \setminus \{0\}$. By Definition 32, if $\phi_w(v_1) = s_1$, $s_1 \geq 1$ then $w = \rightarrow|^{s_1}$ and if $\phi_w(v_1) = s_2$, $s_2 \leq -1$ then $w = \leftarrow|^{-s_2}$. This implies that $w \in ((\rightarrow + \leftarrow)|^+)^+$ for the Induction Basis.

Induction Hypothesis: Assume that G_w has no loops implies $w \in ((\rightarrow + \leftarrow)|^+)^+$ for $\ell(W) \leq n$, $n \geq 1$.

Induction Step: For G_w that has no loops with $\ell(W) = n + 1$, $n \geq 1$. Our aim is to prove that $w \in ((\rightarrow + \leftarrow)|^+)^+$. If $\ell(W) = n + 1$ then $W = v_0e_0 \dots v_n e_n v_{n+1}$ and $\phi_w(v_0) = 0$ since W of G_w has no loops $\phi_w(v_1) \neq 0$ (if $\phi_w(v_1) = 0$ then $v_0 = v_1$ which implies e_0 as the looping edge). So none of consecutive vertices can be mapped to same integer by the PLD ϕ_w that means for all $v_i, v_j \in V_w$: $\phi_w(v_i) \neq \phi_w(v_j)$, $j = i + 1$ where $i, j \geq 0$.

Let us consider some Eulerian trail $X = v_0 e_0 \dots e_{n-1} v_n$ of G_x that has no loops, with the standard PLD ϕ_x ; by Induction Hypothesis, $x \in ((\rightarrow + \leftarrow)|^+)^+$. Let us consider another Eulerian trail $A = v_n e_n v_{n+1}$ of G_a with the standard PLD ϕ_a that has no loops, again by Induction Hypothesis $a \in ((\rightarrow + \leftarrow)|^+)^+$. Here a is as same as w in the Induction Basis.

By Lemma 3, since v_n is the terminus of the Eulerian trail X and origin of the Eulerian trail A we have $X \cdot_{v_n} A$ is also an Eulerian trail of $G_x \cup G_a$. Here $W = X \cdot_{v_n} A$ and $G_w = G_x \cup G_a$. This implies that $w = x \cdot a$ and this implies that $w \in ((\rightarrow + \leftarrow)|^+)^+$. Hence G_w that has no loops with $\ell(W) = n + 1$, $n \geq 1$, implies $w \in ((\rightarrow + \leftarrow)|^+)^+$. \square

Let us define a deterministic finite automaton (DFA) A that accepts $ET_{\text{loop-free}}$, a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where

- $Q = \{q_0, q_1, q_2\}$ is the set of states,
- $\Sigma = \{\rightarrow, \leftarrow, |\}$ is the input alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ as described in the state diagram (Fig. 2.9),
- $q_0 \in Q$ is the initial state,
- $F = \{q_2\} \subseteq Q$ is the set of final states.

The language accepted by A , $L(A) = ET_{\text{loop-free}}$, i. e., $L(A) = \{w \in \Sigma^* : w \in ET \setminus \{\varepsilon\} \wedge \hat{\delta}(q_0, w) = q_2\} = \{w \in \Sigma^* : w \in ET \setminus \{\varepsilon\} \wedge w \in ((\rightarrow + \leftarrow)|^+)^+\}$.

Let us formally prove in Lemma 12 the correctness of the DFA A in Fig. 2.9 accepting the language $ET_{\text{loop-free}}$ which is given by mutual induction.

Lemma 12. *Let A be the machine in Fig. 2.9. Then $L(A) = \{w \in \Sigma^* : w \in ET \setminus \{\varepsilon\} \wedge w \in L(E)\}$, where $E = ((\rightarrow + \leftarrow)|^+)^+$.*

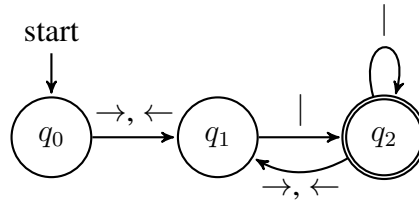


Figure 2.9: DFA A

Proof. The proof is by mutual induction involving the following statements:

1. $\hat{\delta}(q_0, w) = q_1$ if and only if $w \in (\rightarrow + \leftarrow)(|+(\rightarrow + \leftarrow))^*$.
2. $\hat{\delta}(q_0, w) = q_2$ if and only if $w \in (\rightarrow + \leftarrow)(|+(\rightarrow + \leftarrow))^*|^+$. The proof of this statement ensures that the language of the DFA A as described in Fig. 2.9 is the set of strings in ET that does not have the occurrence of only the \rightarrow or \leftarrow arrows which makes it to be loop-free. The proof is by induction on $|h(w)|$, where $h : \Sigma^* \rightarrow \{\rightarrow, \leftarrow\}^*$ such that $h(\rightarrow) = \rightarrow$, $h(\leftarrow) = \leftarrow$ and $h(|) = \varepsilon$.

Induction Basis: If $|h(w)| = 1$, then $w \in \rightarrow|^+$ or $w \in \leftarrow|^+$. Statement (2) says that $\hat{\delta}(q_0, w) = q_2$ which is true. As $w \in \rightarrow|^+$ or $w \in \leftarrow|^+$ does not satisfy the conditions of statement (1), it is clear that the statement (1) is also true.

Induction Hypothesis: Assume for all $w \in ((\rightarrow + \leftarrow)|^+)^+$ with $|h(w)| \leq n$, $n \geq 1$ that both statements (1) and (2) are true.

Induction Step: Consider $w \in ((\rightarrow + \leftarrow)|^+)^+$ with $|h(w)| \leq n + 1$, $n \geq 1$. Our claim is that to show both statements (1) and (2) are true.

Let $w = xa$ with $x \in ((\rightarrow + \leftarrow)|^+)^+$, $|h(x)| = n$ and a needs to start with \rightarrow or \leftarrow . Since $|h(w)| = n + 1$, $n \geq 1$ we have $|h(xa)| = |h(x)| + |h(a)| = n + 1$ which implies that $|h(a)| = 1$. This implies that $a \in (\rightarrow + \leftarrow)|^+$.

1. Suppose $w \in (\rightarrow + \leftarrow)(|+(\rightarrow + \leftarrow))^*$. Since there is a transition from q_0 to q_1 on reading \rightarrow or \leftarrow and there are transitions from q_1 to q_1 on reading $|$'s followed by \rightarrow (or $|$'s followed by \leftarrow) repeatedly it follows $\hat{\delta}(q_0, w) = q_1$. Assume $\hat{\delta}(q_0, w) = q_1$. However this is possible only if w is a word from $(\rightarrow + \leftarrow)(|+(\rightarrow + \leftarrow))^*$.
2. Suppose $w \in (\rightarrow + \leftarrow)(|+(\rightarrow + \leftarrow))^*|^+$. By statement (1) we know that if $w' \in (\rightarrow + \leftarrow)(|+(\rightarrow + \leftarrow))^*$ then $\hat{\delta}(q_0, w') = q_1$. Since there are transitions from q_1 to q_2 on reading $|$'s it follows that $\hat{\delta}(q_0, w) = q_2$. Assume $\hat{\delta}(q_0, w) = q_2$. On examining the transition function we find that $\hat{\delta}(q_0, w) = q_2$ is possible only if $w \in (\rightarrow + \leftarrow)(|+(\rightarrow + \leftarrow))^*|^+$.

□

Theorem 5. $ET_{\text{loop-free}} \in \mathcal{REG}$

Proof. $ET_{\text{loop-free}} = \{w \in \Sigma^* : w \in ET \setminus \{\varepsilon\} \wedge w \in ((\{\rightarrow\} \cup \{\leftarrow\})\{| \}^+)^+\}$ by Lemma 11 and by Lemma 12, $ET_{\text{loop-free}} = L(A)$ for the finite automaton A in Fig. 2.9. Hence, $ET_{\text{loop-free}} \in \mathcal{REG}$. □

Chapter 3

Jumping Finite Automata

It is evident from the history of automata theory, the classical finite automaton has been extended in many different ways: two-way automata, multi-head automata, automata with additional resources (counters, stacks, etc.), and so on. However, for all these variants, it is always the case that the input is read in a continuous fashion. On the other hand, there exist models that are closer to the classical model in terms of computational resources, but that differ in how the input is processed (e. g., restarting automata [90] and biautomata [66]).

One such model that has drawn comparatively little attention is the jumping finite automata (JFA) introduced by Meduna and Zemek [85, 86], in which input head can jump to an arbitrary position within the remaining input after reading and consuming the input symbol.

We introduce a variant of regular-like expressions, called alphabetic shuffle expressions, that characterize JFA languages [32] in terms of expressions using shuffle, union, and iterated shuffle (or shuffle star), which enables us to put them into the context of classical formal language results, especially we put them into the context of earlier literature in the area of shuffle expressions. We show that JFA languages are closed under iterated shuffle which was an open question from [86]. This approach also clarifies the closure properties under Boolean operations.

3.1 JFA and Shuffle Expressions

Let us now recall the language operations of shuffle and permutation, and the notion of semilinearity.

Definition 39. The shuffle operation, denoted by \sqcup , is defined by

$$u \sqcup v = \left\{ x_1 y_1 x_2 y_2 \dots x_n y_n : \begin{array}{l} u = x_1 x_2 \dots x_n, v = y_1 y_2 \dots y_n, \\ x_i, y_i \in \Sigma^*, 1 \leq i \leq n, n \geq 1 \end{array} \right\},$$

$$L_1 \sqcup L_2 = \bigcup_{\substack{u \in L_1 \\ v \in L_2}} (u \sqcup v),$$

for $u, v \in \Sigma^*$ and $L_1, L_2 \subseteq \Sigma^*$.

Note that an inductive definition of shuffle operation, equivalent to Definition 39 is available in [81].

Definition 40. For $L \subseteq \Sigma^*$, the iterated shuffle of L is

$$L^{\sqcup, *} = \bigcup_{n=0}^{\infty} L^{\sqcup, n},$$

where $L^{\sqcup, 0} = \{\varepsilon\}$ and $L^{\sqcup, i} = L^{\sqcup, i-1} \sqcup L$, $i \geq 1$.

The set of permutations of a word can be then conveniently defined using the shuffle operation.

Definition 41. The set $\text{perm}(w)$ of all permutations of w is inductively defined as follows:

$$\text{perm}(w) = \begin{cases} \{\varepsilon\}, & |w| = 0, \\ \{a\} \sqcup \text{perm}(u), & w = a \cdot u, a \in \Sigma, u \in \Sigma^*. \end{cases}$$

The permutation operator extends to languages in the natural way as follows:

$$\text{perm}(L) = \bigcup_{w \in L} \text{perm}(w), L \subseteq \Sigma^*.$$

Observation 4. $\forall x, y \in \Sigma^* : x \in \text{perm}(y) \iff y \in \text{perm}(x)$.

Definition 42. A subset $A \subseteq \mathbb{N}^n$ is said to be linear if there are $v, v_1, \dots, v_m \in \mathbb{N}^n$ such that

$$A = \{v + k_1 v_1 + k_2 v_2 + \dots + k_m v_m : k_1, k_2, \dots, k_m \in \mathbb{N}\}.$$

A subset $A \subseteq \mathbb{N}^n$ is said to be semilinear if it is a finite union of linear sets.

A permutation of the coordinates in \mathbb{N}^n preserves semilinearity. Let Σ be a finite set of n elements. A *Parikh mapping* ψ from Σ^* into \mathbb{N}^n is a mapping defined by first choosing an enumeration a_1, \dots, a_n of the elements of Σ and then defining inductively $\psi(\varepsilon) = (0, \dots, 0)$, $\psi(a_i) = (\delta_{1,i}, \dots, \delta_{n,i})$, where $\delta_{j,i} = 0$ if $i \neq j$ and $\delta_{j,i} = 1$ if $i = j$, and $\psi(au) = \psi(a) + \psi(u)$ for all $a \in \Sigma$, $u \in \Sigma^*$. For clarity, we sometimes add the alphabet Σ as a subscript to ψ . Any two Parikh mappings from Σ^* into \mathbb{N}^n differ only by a permutation of the coordinates of \mathbb{N}^n . Hence, the concept introduced in the following definition is well-defined.

Definition 43. Let Σ be a finite set of n elements. A subset $A \subseteq \Sigma^*$ is said to be a language with the semilinear property, or *slip language* for short, if $\psi(A)$ is a semilinear subset of \mathbb{N}^n for a Parikh mapping ψ of Σ^* into \mathbb{N}^n . The class of all slip languages is denoted by \mathcal{PSL} .

Following Meduna and Zemek [85, 86], we denote a *general finite machine* as $M = (Q, \Sigma, R, s, F)$, where Q is a finite set of *states*, Σ is the *input alphabet*, $\Sigma \cap Q = \emptyset$, R is a finite set of *rules*¹ of the form $py \rightarrow q$, where $p, q \in Q$ and $y \in \Sigma^*$, $s \in Q$ is the *start state* and $F \subseteq Q$ is a set of *final states*. If all rules $py \rightarrow q \in R$ satisfy $|y| \leq 1$, then M is a *finite machine*. We interpret M in two ways.

- As a (general) finite automaton: a *configuration* of M is any string in $Q\Sigma^*$, the binary *move relation* on $Q\Sigma^*$, written as \Rightarrow , is defined as follows:

$$pw \Rightarrow qz \iff \exists py \rightarrow q \in R : w = yz.$$

- As a (general) jumping finite automaton: a *configuration* of M is any string in $\Sigma^*Q\Sigma^*$, the binary *jumping relation* on $\Sigma^*Q\Sigma^*$, written as \curvearrowright , satisfies:

$$vpw \curvearrowright v'qz' \iff \exists py \rightarrow q \in R \exists z \in \Sigma^* : w = yz \wedge vz = v'z'.$$

We hence obtain the following languages from a (general) finite machine M :

$$\begin{aligned} L_{\text{FA}}(M) &= \{w \in \Sigma^* : \exists f \in F : sw \Rightarrow^* f\}, \\ L_{\text{JFA}}(M) &= \{w \in \Sigma^* : \exists u, v \in \Sigma^* \exists f \in F : w = uv \wedge usv \curvearrowright^* f\}. \end{aligned}$$

where the first defines the language classes \mathcal{REG} and the second defines \mathcal{JFA} (accepted by JFAs) and \mathcal{GJFA} (accepted by GJFAs). Moreover, \mathcal{CFL} denotes the class of context-free languages.

Next, we define a special type of expressions that use the shuffle operator. Such shuffle expressions have been an active field of study over decades; we only point the reader to [59], [60] and [61].

¹We also refer to rules as *transitions* with *labels* from Σ^* .

Definition 44. Symbols \emptyset , ε and each $a \in \Sigma$ are shuffle expressions. If S_1, S_2 are shuffle expressions, then $(S_1 \cdot S_2)$, $(S_1 + S_2)$, S_1^* , $(S_1 \sqcup S_2)$ and $S_1^{\sqcup, *}$ are shuffle expressions, and nothing else is a shuffle expression. The language $L(S)$ generated by a shuffle expression S is defined as follows: $L(\emptyset) = \emptyset$, $L(\varepsilon) = \varepsilon$, $L(a) = a$. If $L(S_1) = L_1$ and $L(S_2) = L_2$, then $L((S_1 \cdot S_2)) = L_1 \cdot L_2$, $L((S_1 + S_2)) = L_1 \cup L_2$, $L(S_1^*) = L_1^*$, $L((S_1 \sqcup S_2)) = L_1 \sqcup L_2$, and $L(S_1^{\sqcup, *}) = L_1^{\sqcup, *}$.

We first recall the SHUF expressions introduced by Jantzen [58], from which we then derive α -SHUF expressions, which are tightly linked to jumping finite automata.

Definition 45. Symbols \emptyset, ε and each $w \in \Sigma^+$ are (atomic) SHUF expressions. If S_1, S_2 are SHUF expressions, then $(S_1 + S_2)$, $(S_1 \sqcup S_2)$ and $S_1^{\sqcup, *}$ are SHUF expressions. The semantics of SHUF expressions is defined as follows:

- $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$, $L(w) = \{w\}$, $w \in \Sigma^+$,
- For SHUF expressions S_1 and S_2 ,
 $L(S_1 + S_2) = L(S_1) \cup L(S_2)$,
 $L(S_1 \sqcup S_2) = L(S_1) \sqcup L(S_2)$, and
 $L(S_1^{\sqcup, *}) = L(S_1)^{\sqcup, *}$.

Definition 46. A SHUF expression is an α -SHUF expression, if its atoms are only \emptyset, ε or single symbols $a \in \Sigma$.

Since α -SHUF expressions are SHUF expressions, the semantics follows as it is already defined. Let us illustrate the concepts defined above by two examples.

Example 10. Let M be the finite machine depicted in Figure 3.1, which accepts the regular language $L_{\text{FA}}(M) = L((abc)^*)$. However, if we interpret M as a JFA, it accepts $L_{\text{JFA}}(M) = \{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$. Obviously, $L_{\text{JFA}}(M)$ is also defined by the α -SHUF expression $(a \sqcup b \sqcup c)^{\sqcup, *}$.

Example 11. The general finite machine M' depicted in Figure 3.2 accepts the regular language $L_{\text{FA}}(M') = L((abcd)^*)$. However, it is not easy to describe the language $L_{\text{JFA}}(M')$ in a simple way. Obviously, $\text{perm}(L_{\text{FA}}(M')) \neq L_{\text{JFA}}(M')$ since $bacd \notin L_{\text{JFA}}(M')$ and, furthermore, the SHUF expression $(ab \sqcup cd)^*$ does not describe $L_{\text{JFA}}(M')$ either.

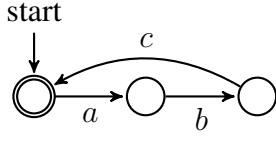


Figure 3.1: Finite Machine M

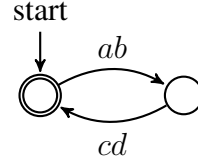


Figure 3.2: General Finite Machine M'

3.2 Algebraic Properties: Shuffle and Permutation

In this section, we state some basic algebraic properties of shuffle, permutation operations. To this end, we first recall the following computation rules for the shuffle operator from [58].

Proposition 6. *Let M_1, M_2, M_3 be arbitrary languages.*

1. $M_1 \sqcup M_2 = M_2 \sqcup M_1$ (*commutative law*),
2. $(M_1 \sqcup M_2) \sqcup M_3 = M_1 \sqcup (M_2 \sqcup M_3)$ (*associative law*),
3. $M_1 \sqcup (M_2 \cup M_3) = M_1 \sqcup M_2 \cup M_1 \sqcup M_3$ (*distributive law*),
4. $(M_1 \cup M_2)^{\sqcup,*} = (M_1)^{\sqcup,*} \sqcup (M_2)^{\sqcup,*}$,
5. $(M_1^{\sqcup,*})^{\sqcup,*} = (M_1)^{\sqcup,*}$,
6. $(M_1 \sqcup M_2^{\sqcup,*})^{\sqcup,*} = (M_1 \sqcup (M_1 \cup M_2)^{\sqcup,*}) \cup \{\varepsilon\}$.

The second, third and fifth rule are also true for (iterated) catenation instead of (iterated) shuffle. This is no coincidence, as we will see. Recall from [79] that a *commutative semiring* is one whose multiplication is commutative.

We can deduce from the first three computation rules the following result.

Proposition 7. $(2^{\Sigma^*}, \cup, \sqcup, \emptyset, \{\varepsilon\})$ is a commutative semiring.

Proof. The proof follows from the following facts:

- $(2^{\Sigma^*}, \cup, \emptyset)$ is a commutative monoid; this is a well-known set-theoretic statement.
- $(2^{\Sigma^*}, \sqcup, \{\varepsilon\})$ is a commutative monoid; this corresponds to the first two computation rules, plus the fact that $\{\varepsilon\}$ is the identity element with respect to the shuffle operation.

- the distributive law was explicitly stated as the third computation rule.

□

We are now discussing some special properties of operator perm in a different (algebraic) viewpoint. Reminiscent of the presentation in [38], there is another way of looking at the permutation operator. Namely, let $w \in \Sigma^n$ be a word of length n , spelled out as $w = a_1 \cdots a_n$ for $a_i \in \Sigma$. Then, $u \in \text{perm}(w)$ if and only if there exists a bijection $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ such that $u = a_{\pi(1)} \cdots a_{\pi(n)}$. In combinatorics, such bijections are known as permutations. This also shows that $|\text{perm}(w)| \leq (|w|)!$. Next, we summarize two important properties of the operator perm in the following two lemmas.

Lemma 13. *Let $\Sigma = \{a_1, \dots, a_n\}$. The set $\{\text{perm}(w) : w \in \Sigma^*\}$ is a partition of Σ^* . There is a natural bijection between this partition and the set of functions $\mathbb{N}^{|\Sigma|}$, given by the Parikh mapping $\psi_\Sigma : \Sigma^* \rightarrow \mathbb{N}^{|\Sigma|}, w \mapsto (|w|_{a_1}, \dots, |w|_{a_n})$. Namely, $\text{perm}(w) = \psi_\Sigma^{-1}(\psi_\Sigma(w))$ for $w \in \Sigma^*$.*

Proof. Let $w = a_1 \cdots a_n$. Then $u \in \text{perm}(w)$ if and only if there exists a bijection $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ such that $u = a_{\pi(1)} \cdots a_{\pi(n)}$. This implies that $u \in \psi_\Sigma^{-1}(|w|_{a_1}, \dots, |w|_{a_n})$ which implies $u \in \psi_\Sigma^{-1}(\psi_\Sigma(a_1 \cdots a_n))$ and which implies $u \in \psi_\Sigma^{-1}(\psi_\Sigma(w))$ for $w \in \Sigma^*$. □

Lemma 14. $\text{perm} : 2^{\Sigma^*} \rightarrow 2^{\Sigma^*}$ is a hull operator.

Proof. We are going to show the three properties separately.

extensive Clearly, if $w \in L$, then $w \in \text{perm}(w) \subseteq \text{perm}(L)$. Hence, $L \subseteq \text{perm}(L)$.

increasing Consider two languages $L_1 \subseteq L_2$. Consider $w \in \text{perm}(L_1) \cap \Sigma^n$. This means that there is a permutation $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$, say, $w = a_1 \cdots a_n$ for some $a_i \in \Sigma$, such that $u = a_{\pi(1)} \cdots a_{\pi(n)}$ for some $u \in L_1$. As $L_1 \subseteq L_2$, $u \in L_2$. Therefore, $w \in \text{perm}(L_2)$.

idempotent Let $w \in \text{perm}(\text{perm}(L)) \cap \Sigma^n$ with $w = a_1 \cdots a_n$ for $a_i \in \Sigma$. This means that there is a permutation $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ such that $u = a_{\pi(1)} \cdots a_{\pi(n)}$ for some $u \in \text{perm}(L)$. This means that there is another permutation π' such that $u' = a_{\pi'(\pi(1))} \cdots a_{\pi'(\pi(n))} \in L$. As the composition of π and π' is again a permutation, we find that $w \in \text{perm}(L)$. Hence, $\text{perm}(\text{perm}(L)) \subseteq \text{perm}(L)$, and as perm is extensive, $\text{perm}(\text{perm}(L)) = \text{perm}(L)$.

This shows the claim. □

Due to the well-known correspondence between hull operators and (systems of) closed sets, we will also speak about *perm-closed languages* in the following, i. e., languages L satisfying $\text{perm}(L) = L$. Such languages are also said to be *commutative*, see [74]. Note that there exists a possibly better known semiring in formal language theory, using catenation instead of shuffle; let us make this explicit in the following statement.

Proposition 8 ([33]). $(2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\varepsilon\})$ is a semiring that is not commutative whenever $|\Sigma| > 1$.

Another algebraic interpretation can be given as follows.

Proposition 9. *Parikh mappings can be interpreted as semiring morphisms from $(2^{\Sigma^*}, \cup, \sqcup, \emptyset, \{\varepsilon\})$ to $(2^{\mathbb{N}^{|\Sigma|}}, \cup, +, \emptyset, \{\vec{0}\})$, where $\vec{0}$ is the tuple with $|\Sigma|$ zeros.*

Proof. The proof follows from the following facts: (i) For $L_1, L_2 \subseteq \Sigma^* : \psi_{\Sigma}(L_1 \cup L_2) = \psi_{\Sigma}(L_1) \cup \psi_{\Sigma}(L_2)$, (ii) For $L_1, L_2 \subseteq \Sigma^* : \psi_{\Sigma}(L_1 \sqcup L_2) = \psi_{\Sigma}(L_1) + \psi_{\Sigma}(L_2)$, and (iii) $\psi_{\Sigma}(\emptyset) = \emptyset$ and $\psi_{\Sigma}(\{\varepsilon\}) = \{\vec{0}\}$, $\vec{0}$ is the tuple with $|\Sigma|$ zeros. \square

Especially, we conclude:

Proposition 10. *For $u, v \in \Sigma^*$, $\text{perm}(u) = \text{perm}(v)$ if and only if $\psi_{\Sigma}(u) = \psi_{\Sigma}(v)$. For $L_1, L_2 \subseteq \Sigma^*$, $\text{perm}(L_1) = \text{perm}(L_2)$ if and only if $\psi_{\Sigma}(L_1) = \psi_{\Sigma}(L_2)$.*

Proof. The proof follows from Lemma 13 and from the fact ψ_{Σ} is surjective. \square

Due to Proposition 10, we can call $u, v \in \Sigma^*$ (and also $L_1, L_2 \subseteq \Sigma^*$) *Parikh-equivalent* or *permutation-equivalent* if $\psi_{\Sigma}(u) = \psi_{\Sigma}(v)$ ($\psi_{\Sigma}(L_1) = \psi_{\Sigma}(L_2)$, respectively). The relation between (iterated) catenation and (iterated) shuffle can now be neatly expressed as follows.

Theorem 11. $\text{perm} : 2^{\Sigma^*} \rightarrow 2^{\Sigma^*}$ is a semiring morphism from the semiring $(2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\varepsilon\})$ to the semiring $(2^{\Sigma^*}, \cup, \sqcup, \emptyset, \{\varepsilon\})$ that also respects the iterated catenation resp. iterated shuffle operation.

Clearly, perm cannot be an isomorphism, since the catenation semiring is not commutative, while the shuffle semiring is, see Proposition 7. The proof of the previous theorem, broken into several statements that are also interesting in their own right, is presented in the following. Notice that in the terminology of Ésik and Kuich [21], Theorem 11 can also be stated as follows: $\text{perm} : 2^{\Sigma^*} \rightarrow 2^{\Sigma^*}$ is a starsemiring morphism from $(2^{\Sigma^*}, \cup, \cdot, *, \emptyset, \{\varepsilon\})$ to $(2^{\Sigma^*}, \cup, \sqcup, \sqcup, *, \emptyset, \{\varepsilon\})$. Some of the required properties are also listed in [80] (without giving a proof).

Lemma 15. $\forall u, v \in \Sigma^* : \text{perm}(u \cdot v) = \text{perm}(u) \sqcup \text{perm}(v)$.

Proof. We prove this lemma by induction on $|u|$. **Induction Basis:** $|u| = 1$. So, $u \in \Sigma$. By Definition 41, $\text{perm}(u \cdot v) = \{u\} \sqcup \text{perm}(v) = \text{perm}(u) \sqcup \text{perm}(v)$. **Induction Hypothesis:** For $u \in \Sigma^n$, $\text{perm}(u \cdot v) = \text{perm}(u) \sqcup \text{perm}(v)$. **Induction Step:** Consider $|u| = n + 1$. Let $u = x_1 x_2 \dots x_{n+1}$, $x_i \in \Sigma$. We now claim that $\text{perm}(x_1 x_2 \dots x_{n+1} \cdot v) = \text{perm}(x_1 x_2 \dots x_{n+1}) \sqcup \text{perm}(v)$.

$$\begin{aligned} \text{perm}(x_1 x_2 \dots x_{n+1} \cdot v) &= \{x_1\} \sqcup \text{perm}(x_2 \dots x_{n+1} \cdot v) \text{ (by Definition 41)} \\ &= \{x_1\} \sqcup \text{perm}(x_2 \dots x_{n+1}) \sqcup \text{perm}(v) \text{ (IH)} \\ &= \text{perm}(x_1 x_2 \dots x_{n+1}) \sqcup \text{perm}(v) \text{ (by Definition 41)}. \end{aligned}$$

Therefore, $\text{perm}(u \cdot v) = \text{perm}(u) \sqcup \text{perm}(v)$. □

Lemma 16. $\forall u, v \in \Sigma^*: u \sqcup v \subseteq \text{perm}(u \cdot v)$.

Proof. It is clear that $u \sqcup v \subseteq \text{perm}(u) \sqcup \text{perm}(v)$, as perm is a hull operator, see Lemma 14, and also by monotonicity $L_1 \subseteq M_1, L_2 \subseteq M_2$ implies $L_1 \sqcup L_2 \subseteq M_1 \sqcup M_2$. According to Lemma 15 $\text{perm}(u) \sqcup \text{perm}(v) = \text{perm}(u \cdot v)$. Therefore, $u \sqcup v \subseteq \text{perm}(u \cdot v)$. □

As a consequence of Lemma 16 and since perm is a hull operator, we obtain the following lemma.

Lemma 17. $\forall u, v \in \Sigma^*: \text{perm}(u \sqcup v) = \text{perm}(u \cdot v) = \text{perm}(u) \sqcup \text{perm}(v)$.

Proof. As indicated, from $u \sqcup v \subseteq \text{perm}(u \cdot v)$ we conclude that $\text{perm}(u \sqcup v) \subseteq \text{perm}(\text{perm}(u \cdot v)) = \text{perm}(u \cdot v)$. Conversely, as $\{u \cdot v\} \subseteq u \sqcup v$, $\text{perm}(u \cdot v) \subseteq \text{perm}(u \sqcup v)$. Hence, $\text{perm}(u \sqcup v) = \text{perm}(u \cdot v)$. Now this together with Lemma 15 gives $\text{perm}(u \sqcup v) = \text{perm}(u) \sqcup \text{perm}(v)$. □

Lemma 18. Let $L, L_1, L_2 \subseteq \Sigma^*$. Then

1. $\text{perm}(L^{n+1}) = \text{perm}(L^n \sqcup L)$,
2. $\text{perm}(L_1) \sqcup \text{perm}(L_2) = \text{perm}(L_1 \sqcup L_2) = \text{perm}(L_1 \cdot L_2)$, and
3. $(\text{perm}(L))^{\sqcup, *} = \text{perm}(L^{\sqcup, *}) = \text{perm}(L^*)$.

Proof. We are going to prove three parts separately.

1. The inclusion $\text{perm}(L^{n+1}) \subseteq \text{perm}(L^n \sqcup L)$ is true, since $L^{n+1} \subseteq L^n \sqcup L$. We now prove the other inclusion $\text{perm}(L^{n+1}) \supseteq \text{perm}(L^n \sqcup L)$. Let $w \in L^n \sqcup L$, then $\exists u \in L^n, v \in L : w \in u \sqcup v$. This implies that $\exists u \in L^n, v \in L : w \in \text{perm}(u \cdot v)$ by Lemma 16. Therefore, $\text{perm}(L^{n+1}) = \text{perm}(L^n \sqcup L)$.

2. Consider $L_1 \subseteq \Sigma^*$, $L_2 \subseteq \Sigma^*$. Let $w \in \text{perm}(L_1) \sqcup \text{perm}(L_2)$. Let $x' \in \text{perm}(L_1)$, $y' \in \text{perm}(L_2)$ such that $w \in x' \sqcup y'$. Hence, there exists some $x \in L_1$ such that $x' \in \text{perm}(x)$ (also $x \in \text{perm}(x')$). Likewise, there exists some $y \in L_2$ with $y' \in \text{perm}(y)$. Hence, $w \in \text{perm}(x) \sqcup \text{perm}(y) = \text{perm}(x \sqcup y)$ by Lemma 17. Therefore, $w \in \text{perm}(L_1 \sqcup L_2) \cap \text{perm}(L_1 \cdot L_2)$. Similarly, if $w \in \text{perm}(L_1 \sqcup L_2)$ then $w \in \text{perm}(L_1) \sqcup \text{perm}(L_2)$. Hence, $\text{perm}(L_1) \sqcup \text{perm}(L_2) = \text{perm}(L_1 \sqcup L_2)$.
3. We will prove $(\text{perm}(L))^{\sqcup, n} = \text{perm}(L^{\sqcup, n})$ and $\text{perm}(L^{\sqcup, n}) = \text{perm}(L^n)$ by induction on n .

Induction Basis: $(\text{perm}(L))^{\sqcup, 0} = \{\varepsilon\} = \text{perm}(\varepsilon) = \text{perm}(L^{\sqcup, 0})$.

Induction Hypothesis: $(\text{perm}(L))^{\sqcup, n} = \text{perm}(L^{\sqcup, n})$.

Induction Step: We now claim that $(\text{perm}(L))^{\sqcup, n+1} = \text{perm}(L^{\sqcup, n+1})$.

$$\begin{aligned}
(\text{perm}(L))^{\sqcup, n+1} &= (\text{perm}(L))^{\sqcup, n} \sqcup \text{perm}(L) \text{ (By Definition 40)} \\
&= \text{perm}(L^{\sqcup, n}) \sqcup \text{perm}(L) \text{ (By Induction Hypothesis)} \\
&= \text{perm}(L^{\sqcup, n} \sqcup L) \text{ (By (2) in Lemma 18)} \\
&= \text{perm}(L^{\sqcup, n+1}) \text{ (By Definition 40)}.
\end{aligned}$$

We now prove $\text{perm}(L^{\sqcup, n+1}) = \text{perm}(L^{n+1})$ by induction on n .

Induction Basis: $\text{perm}(L^{\sqcup, 0}) = \text{perm}(\varepsilon) = \{\varepsilon\} = \text{perm}(\varepsilon) = \text{perm}(L^0)$.

Induction Hypothesis: $\text{perm}(L^{\sqcup, n}) = \text{perm}(L^n)$.

Induction Step: We now claim that $\text{perm}(L^{\sqcup, n+1}) = \text{perm}(L^{n+1})$.

$$\begin{aligned}
\text{perm}(L^{\sqcup, n+1}) &= \text{perm}(L^{\sqcup, n} \sqcup L) \text{ (By Definition 40)} \\
&= \text{perm}(L^{\sqcup, n}) \sqcup \text{perm}(L) \text{ (By (2) in Lemma 18)} \\
&= \text{perm}(L^n) \sqcup \text{perm}(L) \text{ (By Induction Hypothesis)} \\
&= \text{perm}(L^n \sqcup L) \text{ (By (2) in Lemma 18)} \\
&= \text{perm}(L^{n+1}) \text{ (By (1) Lemma 18)}.
\end{aligned}$$

By the definitions of iterated catenation (Kleene star) and iterated shuffle, the claim of the second part follows. \square

Proof of Theorem 11

Proof. Recall that perm , in order to be a semiring morphism, should satisfy the following properties: (i) By an easy standard set-theoretic argument we get $\forall L_1, L_2 \subseteq \Sigma^* : \text{perm}(L_1 \cup L_2) = \text{perm}(L_1) \cup \text{perm}(L_2)$, (ii) By Lemma 18

$\forall L_1, L_2 \subseteq \Sigma^* : \text{perm}(L_1 \cdot L_2) = \text{perm}(L_1) \sqcup \text{perm}(L_2)$, and (iii) $\text{perm}(\emptyset) = \emptyset$ and $\text{perm}(\{\varepsilon\}) = \{\varepsilon\}$ are trivial claims. Furthermore, we claim an according preservation property for the iterated catenation resp. shuffle, which is explicitly stated and proven in Lemma 18. \square

Remark 9. *Let us make some further algebraic consequences explicit.*

(i) $\text{perm}(L)$ can be seen as the canonical representative of all languages \tilde{L} that are permutation-equivalent to L .

(ii) As perm is a morphism, there is in fact a semiring isomorphism between the permutation-closed languages (over Σ) and $\mathbb{N}^{|\Sigma|}$, in this case, which is basically a Parikh mapping.

(iii) There is a further natural isomorphism between the monoid $(\mathbb{N}^{|\Sigma|}, +, \vec{0})$ and the free commutative monoid generated by Σ .

3.3 The Language Class \mathcal{JFA}

Lemma 19. *Let $M = (Q, \Sigma, R, s, F)$ be a finite machine. Then the connection between both rewrite relations defined via M : $\forall p, q \in Q \forall w \in \Sigma^* (\exists u, v \in \Sigma^* : w = uv \text{ and } upv \curvearrowright^* q) \iff (\exists x \in \text{perm}(w) : px \Rightarrow^* q)$.*

Proof. Let $p, q \in Q$ and let $w \in \Sigma^*$, we prove by induction on $|w|$. Induction Basis: $\exists u, v \in \Sigma^* : w = uv$ and $upv \curvearrowright^0 q \iff p = q$ and $u = \varepsilon = v$ and $w = \varepsilon \iff p\varepsilon \Rightarrow^0 q$ and $\varepsilon \in \text{perm}(w)$.

Induction Hypothesis: We assume that the lemma is true for all $w \in \Sigma^* : |w| \leq n$. That is $(\exists u, v \in \Sigma^* : w = uv \text{ and } upv \curvearrowright^n q) \iff (\exists x \in \text{perm}(w) : px \Rightarrow^n q)$.

Induction Step: We now prove that the lemma is true for all $w \in \Sigma^* : |w| = n + 1$.

$$\begin{aligned}
& \exists u, v \in \Sigma^* : w = uv \text{ and } upv \curvearrowright^{n+1} q \\
& \iff \exists u, v \in \Sigma^* : w = uv \exists u', v' \in \Sigma^* \exists r \in Q : py \rightarrow r \in R \\
& \text{and } upv \curvearrowright u'rv' \curvearrowright^n q \\
& \iff \exists u', v' \in \Sigma^* : v = yv', u = u' \text{ and } py \rightarrow r \in R \text{ and } u'rv' \curvearrowright^n q \\
& \iff \exists u', v' \in \Sigma^* : v = yv', u = u' \text{ and } py \rightarrow r \in R \\
& \text{and } \exists x' \in \text{perm}(u'v') : rx' \Rightarrow^n q \text{ (By Induction Hypothesis)} \\
& \iff u', v' \in \Sigma^* : v = yv', u = u' \\
& \text{and } pyx' \Rightarrow rx' \Rightarrow^n q \\
& \iff pyx' \Rightarrow rx' \Rightarrow^n q \text{ and } yx' \in \text{perm}(uv) \\
& \iff px \Rightarrow^{n+1} q \text{ and } x \in \text{perm}(w).
\end{aligned}$$

$yx' \in perm(uv)$ in the last step follows by:

$$\begin{aligned}
perm(yx') &= \{y\} \sqcup perm(x') \text{ (By Definition 41)} \\
&= \{y\} \sqcup perm(wv') \text{ (Since } x' \in perm(u'v')\text{)} \\
&= perm(uyv') \text{ (By Definition 41)} \\
&= perm(uv) \text{ (Since } yv' = v\text{)}.
\end{aligned}$$

$$\exists u, v \in \Sigma^* : w = uv, upv \curvearrowright^{n+1} q \iff \exists x \in perm(w) : px \Rightarrow^{n+1} q. \quad \square$$

Lemma 20. *Let M be a finite machine. Then $L_{JFA}(M) = perm(L_{FA}(M))$.*

Proof. Let $M = (Q, \Sigma, R, s, F)$ be a finite machine. $L_{JFA}(M) = \{w \in \Sigma^* : \exists u, v \in \Sigma^* \exists f \in F : w = uv \wedge usv \curvearrowright^* f\}$. $L_{FA}(M) = \{w \in \Sigma^* : \exists f \in F : sw \Rightarrow^* f\}$. Now our claim is that $L_{JFA}(M) = perm(L_{FA}(M))$.

$$\begin{aligned}
w \in L_{JFA}(M) &\iff \exists u, v \in \Sigma^* w = uv \exists f \in F : usv \curvearrowright^* f \\
&\text{(By the definition of } L_{JFA}(M)\text{)} \\
&\iff \exists u, v \in \Sigma^* w = uv \exists f \in F \exists n \in \mathbb{N} : usv \curvearrowright^n f \\
&\text{(By the definition of reflexive transitive closure of } \curvearrowright : \curvearrowright^* = \bigcup_{n \in \mathbb{N}} \curvearrowright^n\text{)} \\
&\iff \exists x \in perm(w) \exists f \in F \exists n \in \mathbb{N} : sx \Rightarrow_{FA}^n f \text{ (By Lemma 19)} \\
&\iff \exists x \in perm(w) \exists f \in F : sx \Rightarrow_{FA}^* f \\
&\text{(By the definition of reflexive transitive closure of } \Rightarrow : \Rightarrow^* = \bigcup_{n \in \mathbb{N}} \Rightarrow^n\text{)} \\
&\iff \exists x \in perm(w) : x \in L_{FA}(M) \text{ (By the definition of } L_{FA}(M)\text{)} \\
&\iff \exists x \in L_{FA}(M) : w \in perm(x) \text{ (By Observation 4)} \\
&\iff w \in \bigcup_{x \in L_{FA}(M)} perm(x) \text{ (By the definition of union)} \\
&\iff w \in perm(L_{FA}(M)) \text{ (By Definition 41)}.
\end{aligned}$$

□

As an example for this lemma we can recall Example 10 of the \mathcal{JFA} , $L_{JFA}(M) = \{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\} = perm(L_{FA}(M))$, which is nothing but the $perm$ of $L_{FA}(M) = \{abc\}^*$.

Lemma 21. *If R is an α -SHUF expression, then $L(R)$ is a \mathcal{JFA} .*

Proof. The proof is by structural induction. **Basis:** It has three cases, \emptyset, ε and each $a \in \Sigma$. If R is an α -SHUF expression and R is one of \emptyset, ε or a , then $L(R)$ would be $L(\emptyset) = \{\emptyset\}$, $L(\varepsilon) = \{\varepsilon\}$ or $L(a) = \{a\}$, respectively, which are \mathcal{JFA} s, accepted by some finite machine $M = (Q, \Sigma, R, s, F)$. Let R_1, R_2 are α -SHUF expressions such that, $L(R_1), L(R_2)$ are in \mathcal{JFA} , accepted by M_1, M_2 respectively.

$$\begin{aligned}
L((R_1)^{\sqcup,*}) &= (L(R_1))^{\sqcup,*} \text{ (By Definitions 46 and 45)} \\
&= (L_{JFA}(M_1))^{\sqcup,*} \text{ (By Induction)} \\
&= (\text{perm}(L_{FA}(M_1)))^{\sqcup,*} \text{ (By Lemma 20)} \\
&= \text{perm}(L_{FA}(M_1)^*) \text{ (By Theorem 18)} \\
&= \text{perm}(L_{FA}(M')) \text{ for some finite machine } M' \\
&\text{(Since regular languages are closed under *)} \\
&= L_{JFA}(M') \text{ (By Lemma 20)}
\end{aligned}$$

The \cup, \sqcup can be proved by the closure properties of \mathcal{JFA} s as in Theorems 26 and 27 (see Section 4.3) of [85].

$$\begin{aligned}
L(R_1 + R_2) &= L(R_1) \cup L(R_2) \text{ (By Definitions 46 and 45)} \\
&= L_{JFA}(M_1) \cup L_{JFA}(M_2) \text{ (By Induction)} \\
&= L_{JFA}(\widetilde{M}) \text{ for some } \widetilde{M} \text{ (Since } \mathcal{JFA}\text{s are closed under } \cup \text{ [85])}.
\end{aligned}$$

$$\begin{aligned}
L(R_1 \sqcup R_2) &= L(R_1) \sqcup L(R_2) \text{ (By Definition 46 and 45)} \\
&= L_{JFA}(M_1) \sqcup L_{JFA}(M_2) \text{ (By Induction)} \\
&= L_{JFA}(\widetilde{M}) \text{ for some } \widetilde{M} \text{ (Since } \mathcal{JFA}\text{s are closed under } \sqcup \text{ [85])}.
\end{aligned}$$

□

By the definition of a jumping finite automaton M , it is clear that $w \in L_{JFA}(M)$ implies that $\text{perm}(w) \subseteq L_{JFA}(M)$, i. e., $\text{perm}(L_{JFA}(M)) \subseteq L_{JFA}(M)$. Since perm is extensive as a hull operator (see Lemma 14), we can conclude:

Corollary 3. *If $L \in \mathcal{JFA}$, then L is perm-closed.*

This also follows from results of [85]. In particular, we mention the following important characterization theorem from [86], that we enrich by combining it with the well-known theorem of Parikh [92] using Proposition 10.

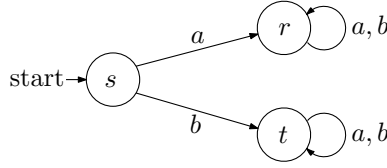


Figure 3.3: An example JFA, final states not specified.

Theorem 12. $\mathcal{JFA} = \text{perm}(\mathcal{REG}) = \text{perm}(\mathcal{CFL}) = \text{perm}(\mathcal{PSL})$.

This theorem also generalizes the main result of [75]. It also indicates certain properties of this language class have been previously derived under different names; for instance, Latteux [74] writes \mathcal{JFA} as $\text{c}(\text{RAT})$, and he mentions yet another characterization for this class in the literature, which is the class of all perm-closed languages whose Parikh image is semilinear; as in Section 3.1, the class of languages whose Parikh image is semilinear is known as *slip languages* [45], or \mathcal{PSL} for short. Due to Lemma 13, there is a natural bijection between \mathcal{JFA} and the recognizable subsets of the monoid $(\mathbb{N}^{|\Sigma|}, +, \vec{0})$. Let us mention one corollary that can be deduced from these connections; for proofs, we refer to [20, 44].

Corollary 4. \mathcal{JFA} is closed under intersection and under complementation.

Notice that the proof given in Theorem 17.4.6 in [86] is wrong, as nondeterminism inherent in JFAs due to the jumping feature is neglected. For instance, consider the deterministic² JFA $M = (\{r, s, t\}, \{a, b\}, R, \{s\}, F)$ with rules according to Figure 3.3. If $F = \{r\}$, then M accepts all words that contain at least one a . But, if $F = \{s, t\}$, then M accepts ε and all words that contain at least one b . This clearly shows that the standard state complementation technique does not work for JFAs.

The Parikh’s theorem in [22] links JFAs to the literature on “commutative context-free languages” for which we give just two references [7, 68]. Also, a sort of normal forms for language classes \mathcal{L} such that $\text{perm}(\mathcal{L}) = \mathcal{JFA}$ have been studied, for instance, the class \mathcal{L} of letter-bounded languages can be characterized in various ways, see [12, 14, 51] for a kind of survey. Since finite languages are regular, we can conclude the following corollary of Theorem 12.

Corollary 5. Let L be a finite language. Then, $L \in \mathcal{JFA}$ if and only if L is perm-closed.

²According to [86], a JFA is *deterministic* if each state has exactly one outgoing transition for each letter.

This also shows that all finite JFA languages are so-called commutative regular languages as studied by Ehrenfeucht, Haussler and Rozenberg in [19]. We will come back to this issue later. Next, we shall show that \mathcal{JFA} coincides with the class of α -SHUF expressions. To this end, we observe that a regular expression E can be easily turned into an α -SHUF expression describing $\text{perm}(L(E))$ by replacing catenations and Kleene stars with shuffles and iterated shuffles (this is a direct consequence of the fact that the perm operator is a semiring morphism as stated in Theorem 11).

Lemma 22. *Let R' be a regular expression. Let the α -SHUF expression R be obtained from R' by consequently replacing all \cdot by \sqcup , and all $*$ by $\sqcup, *$ in R' . Then, $\text{perm}(L(R')) = L(R)$.*

Proof. Let R' be a regular expression. By definition, this means that $L(R') = K$, where K is some expression over the languages \emptyset , $\{\varepsilon\}$ and $\{a\}$, $a \in \Sigma$, using only union, catenation and Kleene-star. By Theorem 11, $\text{perm}(K)$ can be transformed into an equivalent expression K' using only union, shuffle and iterated shuffle. Furthermore, in K' , the operation perm only applies to languages of the form \emptyset , $\{\varepsilon\}$ and $\{a\}$, $a \in \Sigma$, which means that by simply removing all perm operators, we obtain an equivalent expression K'' over languages \emptyset , $\{\varepsilon\}$ and $\{a\}$, $a \in \Sigma$, using only union, shuffle and iterated shuffle. This expression directly translates into the α -SHUF expression R with $L(R) = \text{perm}(L(R'))$. \square

We are now ready to prove our characterization theorem for \mathcal{JFA} .

Theorem 13. *A language $L \subseteq \Sigma^*$ is in \mathcal{JFA} if and only if there is some α -SHUF expression R such that $L = L(R)$.*

Proof. If $L \in \mathcal{JFA}$, then there exists a regular language L' such that $L = \text{perm}(L')$ by Theorem 12. L' can be described by some regular expression R' . By Lemma 22, we find an α -SHUF expression R with $L = \text{perm}(L(R')) = L(R)$.

Conversely, if L is described by some α -SHUF expression R , i. e., $L = L(R)$, then construct the regular expression R' by consequently replacing all \sqcup by \cdot and all $\sqcup, *$ by $*$ in R . Clearly, we face the situation described in Lemma 22, so that we conclude that $\text{perm}(L(R')) = L(R) = L$. As $L(R')$ is a regular language, $\text{perm}(L(R')) = L \in \mathcal{JFA}$ by Theorem 12. \square

As a consequence of Theorem 13, we obtain the following corollary, adding to the list of closure properties given in [85]. Also we can observe that this is true because α -SHUF languages are closed under iterated shuffle.

Corollary 6. *\mathcal{JFA} is closed under iterated shuffle.*

Let us finally mention a second characterization of the finite perm-closed sets in terms of α -SHUF expressions (recall Corollary 5 that states first characterization).

Proposition 14. *Let L be a language. Then, L is finite and perm-closed if and only if there is an α -SHUF expression R , with $L = L(R)$, that does not contain the iterated shuffle operator.*

Proof. Let L be a finite language with $L = \text{perm}(L)$. R_L is a regular expression, with $L(R_L) = L$, which uses only catenation, union operations. As L is perm-closed, α -SHUF expression R obtained from R_L by replacing all catenation by shuffle operators satisfies $L(R) = \text{perm}(L(R_L)) = L$ by Lemma 22 and does not contain the iterated shuffle operator. Conversely, let R be an α -SHUF expression that does not contain the iterated shuffle operator. By combining Theorem 13 with Corollary 3, we know that $L(R)$ is perm-closed. It is rather straightforward that $L(R)$ is also finite. \square

Example 12. *Let M be the finite machine presented in Fig. 3.4. In the standard way, we can turn M into the regular expression*

$$E = ((ab^*ab)^*((ab^*aa) + b)(ab^*aa)^*((ab^*ab) + b))^* \\ (ab^*ab)^*((ab^*aa) + b)(ab^*aa)^*$$

with $L_{\text{FA}}(M) = L(E)$. By Lemma 22, $L_{\text{JFA}}(M) = L(E')$, where

$$E' = ((a \sqcup b^{\sqcup,*} \sqcup a \sqcup b)^{\sqcup,*} \sqcup ((a \sqcup b^{\sqcup,*} \sqcup a \sqcup a) + b) \\ \sqcup (a \sqcup b^{\sqcup,*} \sqcup a \sqcup a)^{\sqcup,*} \sqcup ((a \sqcup b^{\sqcup,*} \sqcup a \sqcup b) + b))^{\sqcup,*} \\ \sqcup (a \sqcup b^{\sqcup,*} \sqcup a \sqcup b)^{\sqcup,*} \sqcup ((a \sqcup b^{\sqcup,*} \sqcup a \sqcup a) + b) \\ \sqcup (a \sqcup b^{\sqcup,*} \sqcup a \sqcup a)^{\sqcup,*} .$$

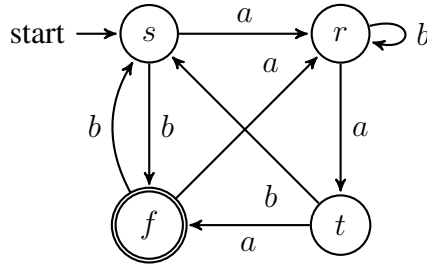


Figure 3.4: The finite machine of Example 12.

3.4 The Language Classes \mathcal{GJFA} and \mathcal{SHUF}

In the previous section, we saw that JFA and α -SHUF expressions correspond to each other in a very similar way as classical regular expressions correspond to finite automata. More precisely, in the translation between α -SHUF expressions and JFA, the atoms of the α -SHUF expression will become the labels of the JFA and vice versa.

GJFAs differ from JFAs only in that the labels can be arbitrary words instead of symbols and, similarly, SHUF expressions differ from α -SHUF expressions only in that the atoms can be arbitrary words. This suggests that a similar translation between GJFAs and SHUF expressions exists and, thus, these devices describe the same class of languages. Unfortunately, this is not the case, which can be demonstrated with a simple example: let $M = (\{s\}, \{a, b, c, d\}, \{sab \rightarrow s, scd \rightarrow s\}, s, \{s\})$ be a GJFA, which naturally translates into the SHUF expression $E = (ab + cd)^{\sqcup, *}$. It can be easily verified that every word that is accepted by M can also be generated by E , but, as $acbd \in (L(E) \setminus L_{\text{JFA}}(M))$, we have $L_{\text{JFA}}(M) \subsetneq L(E)$. In the following, we shall see that not only this naive translation between GJFA and SHUF expressions fails, but the language classes \mathcal{GJFA} and \mathcal{SHUF} are incomparable.

Lemma 23. *Let $M = (\{s\}, \{a, b, c, d\}, \{sab \rightarrow s, scd \rightarrow s\}, s, \{s\})$. Then, $L(M)$ is not a SHUF language.*

Proof. Assume the contrary, let E be a SHUF expression with $L(E) = L(M)$. As the number of occurrences of both a and d are unbounded in words from $L(M)$, one of the two cases must hold:

Case 1: E contains a subexpression $(R)^{\sqcup, *}$ such that there exists a $w \in L(R)$ with $|w|_a \geq 1$ and $|w|_d \geq 1$.

Case 2: E contains a subexpression $R_1 \sqcup R_2$ such that there exists a $w \in L(R_1)$ with $|w|_a \geq 1$ and a $w' \in L(R_2)$ with $|w'|_d \geq 1$.

Both cases imply $L(E)$ contains a word with factor ad . This gives a contradiction, since such words are not in $L_{\text{JFA}}(M)$. \square

Lemma 24. *Let $L = L(ac \sqcup (bd)^{\sqcup, *})$. L is not accepted by any GJFA.*

Proof. Assume the contrary that L is accepted by a GJFA M . Let n be greater than the maximum length of a transition label in M and let $w = ab^n cd^n$. The accepting computation of M on w uses exactly one transition with a label u that contains c .

- If $u = b^i c d^j$ for $i, j \geq 0$, all earlier transitions only consume factors that are completely contained in the prefix ab^{n-i} or the suffix d^{n-j} of $w = ab^{n-i}(b^i c d^j)d^{n-j}$. This implies that, by using the same sequence of transitions, M can accept $w' = b^i c d^j ab^{n-i} d^{n-j}$.
- Otherwise, $u = ab^r c d^s$ for $r, s \geq 0$, i. e., it contains both a and c . By the choice of n , an earlier transition labeled with b^k with $k > 0$ was used. However, this implies that also $w'' = ab^{n-k} c d^n b^k$ is accepted by M .

The case of $w' \in L(M)$ violates the condition that the symbol a precedes c in words from L , while the case of $w'' \in L_{\text{JFA}}(M)$ contradicts the fact that the words in L do not end with b . \square

Lemma 25. $\{ab\}^{\sqcup,*} \in (\mathcal{GJFA} \cap \mathcal{SHUF}) \setminus \mathcal{JFA}$.

Proof. Obviously, $\{ab\}^{\sqcup,*} = L((ab)^{\sqcup,*})$. Furthermore, $\{ab\}^{\sqcup,*} = L_{\text{JFA}}(M)$, where M is the GJFA with a single state s , which is both initial and final, and a single rule $sab \rightarrow s$. As $ab \in \{ab\}^{\sqcup,*}$, but $ba \notin \{ab\}^{\sqcup,*}$, $\{ab\}^{\sqcup,*}$ is not perm-closed, and hence not a JFA language. \square

It is interesting to note that if the permutation closures of separating languages from Lemmas 23 and 24 are taken, then we get JFA languages. As shall be demonstrated next (see Theorem 15), this property holds for all SHUF and GJFA languages.

Theorem 15. $\text{perm}(\mathcal{GJFA}) = \text{perm}(\mathcal{SHUF}) = \text{perm}(\mathcal{PSL}) = \mathcal{JFA}$.

Proof. By Theorem 12, $\mathcal{JFA} = \text{perm}(\mathcal{PSL})$. Let us prove that $\text{perm}(\mathcal{GJFA}) \cap \text{perm}(\mathcal{SHUF}) = \mathcal{JFA}$. Clearly $\mathcal{JFA} \subseteq \text{perm}(\mathcal{GJFA}) \cap \text{perm}(\mathcal{SHUF})$ and we have to show that $\text{perm}(\mathcal{GJFA}) \cap \text{perm}(\mathcal{SHUF}) \subseteq \mathcal{JFA}$.

Case 1: Let $L \in \mathcal{SHUF}$ be described by a SHUF expression X . Then $\text{perm}(L)$ is described by the α -SHUF expression X' that is obtained from X by replacing each atomic word $a_1 \cdots a_n \in \Sigma^*$ of length $n \geq 2$ by the α -SHUF subexpression $a_1 \sqcup \cdots \sqcup a_n$. By Theorem 11, $\text{perm}(L) = \text{perm}(L(X)) = L(X')$.

Case 2: Let $L \in \mathcal{GJFA}$. The well-known construction of a finite automaton that simulates a given general finite automaton can be applied to obtain, from a given GJFA M , a JFA M' with the property $\text{perm}(L_{\text{JFA}}(M)) = L_{\text{JFA}}(M')$.

The correctness of this method immediately follows from our reasoning towards Theorem 12. In both the cases we conclude that $\text{perm}(L)$ lies in \mathcal{JFA} .

Let us now prove that $\text{perm}(\mathcal{GJFA}) = \text{perm}(\mathcal{SHUF}) = \mathcal{JFA}$. Let $L \subseteq \Sigma^*$. Then, the following three claims are equivalent: (i) $L \in \mathcal{JFA}$, (ii) L is perm-closed and $L \in \mathcal{GJFA}$, and (iii) L is perm-closed and $L \in \mathcal{SHUF}$.

As each $L \in \mathcal{JFA}$ is perm-closed and in $\mathcal{GJFA} \cap \mathcal{SHUF}$, we only have to show the upward implications. If $L \in \mathcal{GJFA}$, then by Case 2, $\text{perm}(L) \in \mathcal{JFA}$. If, in addition, L is perm-closed, then $\text{perm}(L) = L$, which shows the claim. Similarly, we can show that, if L is perm-closed and $L \in \mathcal{SHUF}$, then $L \in \mathcal{JFA}$. \square

We summarize the inclusion relations between the language families considered in this chapter in Figure 3.5. In this figure, an arrow from class A to B represents the strict inclusion $A \subsetneq B$. A missing connection between a pair of language families means incomparability.

Theorem 16. *The inclusion and incomparability relations displayed in Figure 3.5 are correct.*

Proof. We first show the correctness of the subset relations. The class $\mathcal{REG} \cap \mathcal{JFA}$ is obviously included in both \mathcal{REG} and \mathcal{JFA} , and any non-commutative regular language and the non-regular JFA language $L = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$ show these subset relations to be proper. That $\mathcal{JFA} \subsetneq \mathcal{SHUF} \cap \mathcal{GJFA}$ follows by definition and Lemma 25. Similarly, both $\mathcal{SHUF} \cap \mathcal{GJFA} \subsetneq \mathcal{GJFA}$ and $\mathcal{SHUF} \cap \mathcal{GJFA} \subsetneq \mathcal{SHUF}$ follows by definition and Lemmas 23 and 24, respectively.

Theorem 15 together with Lemma 14 (as the operator perm is extensive) shows that the classes \mathcal{SHUF} and \mathcal{GJFA} are contained in \mathcal{PSL} . Namely, if this would not be the case, then there should be a language L , say, in $\mathcal{GJFA} \setminus \mathcal{PSL}$. Now, $\text{perm}(L) \in \text{perm}(\mathcal{GJFA}) \subseteq \mathcal{PSL}$, but $L \in \mathcal{PSL}$ if and only if $\text{perm}(L) \in \mathcal{PSL}$ by definition of \mathcal{PSL} , yielding a contradiction.

Similarly, there should be a language L , say, in $\mathcal{SHUF} \setminus \mathcal{PSL}$. Now, $\text{perm}(L) \in \text{perm}(\mathcal{SHUF}) \subseteq \mathcal{PSL}$, but $L \in \mathcal{PSL}$ if and only if $\text{perm}(L) \in \mathcal{PSL}$, again yielding a contradiction. Since \mathcal{SHUF} and \mathcal{GJFA} are incomparable, these two inclusions are proper.

By Parikh's theorem [92] and as the context-free languages do not contain the language studied in Example 10, $\mathcal{CFL} \subsetneq \mathcal{PSL}$. Finally, $\mathcal{REG} \subsetneq \mathcal{CFL}$ is well-known; thus, all the claimed proper subset relations hold. Since \mathcal{JFA} is a proper superclass of $\mathcal{REG} \cap \mathcal{JFA}$, it contains a language not in \mathcal{REG} . Furthermore, as in [86, Lemma 17.3.2], the regular language $\{a\}^*\{b\}^*$ is not in \mathcal{GJFA} .

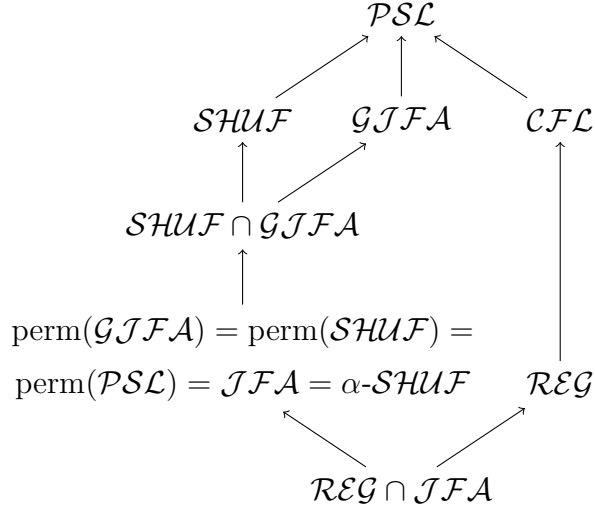


Figure 3.5: Inclusion diagram of our language families.

By a similar argument as used in the proof of Lemma 23, it can also be shown that $\{a\}^*\{b\}^* \notin \mathcal{SHUF}$ (more precisely, since it is a language which is infinite, either a subexpression that contains both a and b is subject to an iterated shuffle operation or two subexpressions which produce only a and b , respectively, are connected by a shuffle operation).

Hence, \mathcal{REG} is incomparable with all the classes on the left side of the diagram. The language of Example 10 is in \mathcal{JFA} , but not in \mathcal{CFL} . Furthermore, $\{a\}^*\{b\}^*$ is a context-free language, which implies that \mathcal{CFL} is also incomparable with all the classes on the left side of the diagram. Finally, the incomparability of the classes \mathcal{SHUF} and \mathcal{GJFA} is established by Lemmas 23, 24 and 25. This concludes the proof. \square

Theorem 17. \mathcal{GJFA} is closed under mirror image.

Proof. Given $L \in \mathcal{GJFA}$. Then there exists a GJFA $M = (Q, \Sigma, R, s, F)$ that accepts L . We have to show that the language $L^R \in \mathcal{GJFA}$. Let us build a GJFA $M^R = (Q, \Sigma, R', s, F)$ that accepts L^R , where R' has the following rule $py^R \rightarrow q$, where $py \rightarrow q \in R \forall p, q \in Q, y \in \Sigma^*$. Now, our claim is that $(L_{JFA}(M))^R = L_{JFA}(M^R)$. Let $w \in L_{JFA}(M)$. Then $w^R \in (L_{JFA}(M))^R$. Now, our claim is that $w^R \in (L_{JFA}(M))^R \iff w^R \in L_{JFA}(M^R)$. i. e., $\forall t, f \in Q \exists u^R, v^R \in \Sigma^* : w^R = v^R u^R \wedge v^R t u^R \curvearrowright_{M^R}^* f \iff \exists u, v \in \Sigma^* : w = uv \wedge utv \curvearrowright_M^* f$. i. e., $\forall t, f \in Q \exists n \in \mathbb{N} \exists u^R, v^R \in \Sigma^* : w^R = v^R u^R \wedge v^R t u^R \curvearrowright_{M^R}^n f \iff \exists u, v \in \Sigma^* : w = uv \wedge utv \curvearrowright_M^n f$. We prove this by induction on the

number of jumps n . **Induction Basis:** When $n = 1$

$$\begin{aligned}
& \exists u^R, v^R \in \Sigma^* : w^R = v^R u^R \wedge v^R t u^R \curvearrowright_{MR}^1 f \\
& \iff \exists u^R, v^R \in \Sigma^* : w^R = v^R u^R \wedge \exists t w^R \rightarrow f \in R' \\
& \iff \exists u, v \in \Sigma^* : w = uv \wedge \exists t w \rightarrow f \in R \\
& \iff \exists u, v \in \Sigma^* : w = uv \wedge utv \curvearrowright_M^1 f.
\end{aligned}$$

Induction Hypothesis: $\forall t, f \in Q \exists n \in \mathbb{N}$

$$\begin{aligned}
& \exists u^R, v^R \in \Sigma^* : w^R = v^R u^R \wedge v^R t u^R \curvearrowright_{MR}^{n-1} f \\
& \iff \exists u, v \in \Sigma^* : w = uv \wedge utv \curvearrowright_M^{n-1} f.
\end{aligned}$$

Induction Step: $\forall t, f \in Q \exists n \in \mathbb{N}$

$$\begin{aligned}
& \exists u^R, v^R \in \Sigma^* : w^R = v^R u^R \wedge v^R t u^R \curvearrowright_{MR}^n f \\
& \iff \exists u^R, v^R, u'^R, v'^R \in \Sigma^* : w^R = v^R u^R \exists r \in Q : ty^R \rightarrow r \in R' \\
& \text{and } v^R t u^R \curvearrowright_{MR}^1 v'^R r u'^R \curvearrowright_{MR}^{n-1} f \\
& \iff \exists u^R, v^R, u'^R, v'^R \in \Sigma^* : w^R = v^R u^R \\
& \text{and } u^R = y^R u'^R \wedge v^R = v'^R \text{ and } ty^R \rightarrow r \in R' \\
& \text{and } v'^R r u'^R \curvearrowright_{MR}^{n-1} f \\
& \iff \exists u, v \in \Sigma^* : w = uv \exists u', v' \in \Sigma^* : u = u' \wedge v = yv' \\
& \text{and } ty \rightarrow r \in R \text{ and } u'rv' \curvearrowright_M^{n-1} f \text{ (By Induction Hypothesis)} \\
& \iff \exists u, v, u', v' \in \Sigma^* \exists r \in Q : ty \rightarrow r \in R \\
& \text{and } utv \curvearrowright_M^1 u'rv' \curvearrowright_M^{n-1} f \\
& \iff \exists u, v \in \Sigma^* : w = uv \wedge utv \curvearrowright_M^n f.
\end{aligned}$$

□

Let us now recall the definitions of \mathcal{SE} and \mathcal{SHUF} from [59]. The family \mathcal{SE} is equal to the class of languages definable by shuffle expressions. \mathcal{SHUF} is a subclass of \mathcal{SE} and has been studied in [58]. Next we prove that \mathcal{SHUF} is closed under mirror image, for this we first prove the following lemmas.

Lemma 26. $(ua \sqcup vb) = (u \sqcup vb)a \cup (ua \sqcup v)b$.

Proof. The proof is based on induction.

Induction Basis: It has two cases, when $u = \varepsilon$ and $v = \varepsilon$. **Case 1:** If $u = \varepsilon$, then $ua \sqcup vb = \varepsilon \cdot a \sqcup vb = a \sqcup vb$. We now prove this by induction on length of v .

Induction Basis for Case 1: When $v = \varepsilon$.

$$\begin{aligned}
a \sqcup vb &= a \sqcup \varepsilon \cdot b \text{ (Since } v = \varepsilon\text{)} \\
&= a \sqcup b \text{ (Since } \varepsilon \cdot b = b\text{)} \\
&= ba \cup ab \text{ (By Definition 39)} \\
&= \{b\}a \cup \{a\}b \text{ (Since } L(b) = \{b\} \text{ and } L(a) = \{a\}\text{)} \\
&= (\varepsilon \sqcup b)a \cup (a \sqcup \varepsilon)b \text{ (By Definition 39)} \\
&= (\varepsilon \sqcup \varepsilon \cdot b)a \cup (\varepsilon \cdot a \sqcup \varepsilon)b \text{ (Since } \varepsilon \cdot a = a \text{ and } \varepsilon \cdot b = b\text{)}.
\end{aligned}$$

Therefore, $(ua \sqcup vb) = (u \sqcup vb)a \cup (ua \sqcup v)b$ for the basis.

Induction Step for Case 1: Let $v = b'v'$.

$$\begin{aligned}
a \sqcup vb &= a \sqcup b'v'b \text{ (Since } v = b'v'\text{)} \\
&= a(b'v'b) \cup b'(a \sqcup v'b) \text{ (By Definition 39)} \\
&= ab'v'b \cup b'((v'b)a \cup (a \sqcup v')b) \text{ (By Induction Hypothesis)} \\
&= ab'v'b \cup b'v'ba \cup b'(a \sqcup v')b \text{ (By Distributive Law)} \\
&= b'v'ba \cup (ab'v' \cup b'(a \sqcup v'))b \text{ (By Distributive Law)} \\
&= b'v'ba \cup (a \sqcup b'v')b \text{ (By Definition 39)} \\
&= vba \cup (a \sqcup v)b \text{ (Since } v = b'v'\text{)}.
\end{aligned}$$

Therefore, $a \sqcup vb = vba \cup (a \sqcup v)b$. Hence the basis for Case 1.

Case 2: If $v = \varepsilon$, then $ua \sqcup vb = ua \sqcup \varepsilon \cdot b = ua \sqcup b$. We now prove this by induction on length of u . **Induction Basis for Case 2:** When $u = \varepsilon$.

$$\begin{aligned}
ua \sqcup b &= \varepsilon \cdot a \sqcup b \text{ (Since } u = \varepsilon\text{)} \\
&= a \sqcup b \text{ (Since } \varepsilon \cdot a = a\text{)} \\
&= ba \cup ab \text{ (By Definition 39)} \\
&= \{b\}a \cup \{a\}b \text{ (Since } L(b) = \{b\} \text{ and } L(a) = \{a\}\text{)} \\
&= (\varepsilon \sqcup b)a \cup (a \sqcup \varepsilon)b \text{ (By Definition 39)} \\
&= (\varepsilon \sqcup \varepsilon \cdot b)a \cup (\varepsilon \cdot a \sqcup \varepsilon)b \text{ (Since } \varepsilon \cdot a = a \text{ and } \varepsilon \cdot b = b\text{)}.
\end{aligned}$$

Therefore, $(ua \sqcup vb) = (u \sqcup vb)a \cup (ua \sqcup v)b$ for the basis.

Induction Step for Case 2: Let $u = a'u'$.

$$\begin{aligned}
ua \sqcup b &= a'u'a \sqcup b \text{ (Since } u = a'u') \\
&= a'(u'a \sqcup b) \cup b(a'u'a) \text{ (By Definition 39)} \\
&= a'((u' \sqcup b)a \cup (u'a)b) \cup ba'u'a \text{ (By Induction Hypothesis)} \\
&= a'(u' \sqcup b)a \cup a'u'ab \cup ba'u'a \text{ (By Distributive Law)} \\
&= (a'(u' \sqcup b) \cup ba'u')a \cup a'u'ab \text{ (By Distributive Law)} \\
&= (a'u' \sqcup b)a \cup a'u'ab \text{ (By Definition 39)} \\
&= (u \sqcup b)a \cup uab \text{ (Since } u = a'u').
\end{aligned}$$

Therefore, $ua \sqcup b = (u \sqcup b)a \cup uab$. Hence the basis for Case 2.

Induction Step: Let $u = a'u'$ and $v = b'v'$.

$$\begin{aligned}
ua \sqcup vb &= a'u'a \sqcup b'v'b \text{ (Since } u = a'u' \text{ and } v = b'v') \\
&= a'(u'a \sqcup b'v'b) \cup b'(a'u'a \sqcup v'b) \text{ (By Definition 39)} \\
&= a'((u' \sqcup b'v'b)a \cup (u'a \sqcup b'v')b) \cup b'((a'u' \sqcup v'b)a \cup (a'u'a \sqcup v')b) \\
&\text{(By Induction Hypothesis)} \\
&= a'(u' \sqcup b'v'b)a \cup a'(u'a \sqcup b'v')b \cup b'(a'u' \sqcup v'b)a \cup b'(a'u'a \sqcup v')b \\
&\text{(By Distributive Law)} \\
&= (a'(u' \sqcup b'v'b) \cup b'(a'u' \sqcup v'b))a \cup (a'(u'a \sqcup b'v') \cup b'(a'u'a \sqcup v'))b \\
&\text{(By Distributive Law)} \\
&= (a'u' \sqcup b'v'b)a \cup (a'u'a \sqcup b'v')b \text{ (By Definition 39)} \\
&= (u \sqcup vb)a \cup (ua \sqcup v)b \text{ (Since } u = a'u' \text{ and } v = b'v').
\end{aligned}$$

Hence, $(ua \sqcup vb) = (u \sqcup vb)a \cup (ua \sqcup v)b$. □

Lemma 27. a) $\forall L_1 \subseteq \Sigma^* L_2 \subseteq \Sigma^* : (L_1 \cup L_2)^R = (L_1)^R \cup (L_2)^R$.
b) $(\bigcup_{i=1}^{\infty} L_i)^R = \bigcup_{i=1}^{\infty} (L_i)^R$.

Proof. a) Let $w \in L_1 \cup L_2$. Then $w \in L_1 \vee w \in L_2$. This implies $w^R \in (L_1)^R \vee w^R \in (L_2)^R$. **Claim:** $(L_1 \cup L_2)^R = (L_1)^R \cup (L_2)^R$. Let $w^R \in (L_1)^R \cup (L_2)^R$.

$$\begin{aligned}
w^R \in (L_1)^R \cup (L_2)^R &\iff w^R \in (L_1)^R \vee w^R \in (L_2)^R \text{ (By Definition of Union)} \\
&\iff w \in L_1 \vee w \in L_2 \text{ (By Definition of Reversal)} \\
&\iff w \in L_1 \cup L_2 \text{ (By Definition of Union)} \\
&\iff w^R \in (L_1 \cup L_2)^R \text{ (By Definition of Reversal)}.
\end{aligned}$$

Hence $(L_1 \cup L_2)^R = (L_1)^R \cup (L_2)^R$. Hence the proof of (a).

b) We prove this lemma by induction on i . **Induction Basis:** When $i = 1$. Then $(L_1)^R = L_1^R$. **Induction Hypothesis:** $(L_1 \cup L_2 \cup \dots \cup L_n)^R = (L_1)^R \cup (L_2)^R \cup \dots \cup (L_n)^R$. **Induction Step:** Let $w \in \bigcup_{i=1}^{\infty} (L_i)$. Then $w \in L_1 \vee w \in L_2 \vee \dots \vee w \in L_n \vee \dots$ that implies $w^R \in (L_1)^R \vee w^R \in (L_2)^R \vee \dots \vee w^R \in (L_n)^R \vee \dots$. Now our Claim is that $(\bigcup_{i=1}^{\infty} L_i)^R = \bigcup_{i=1}^{\infty} (L_i)^R$. Let $w^R \in (\bigcup_{i=1}^{\infty} L_i)^R$. Then we have

$$\begin{aligned} w^R \in (\bigcup_{i=1}^{\infty} L_i)^R &\iff w \in \bigcup_{i=1}^{\infty} L_i \text{ (By Definition of Reversal)} \\ &\iff \exists i \geq 1 : w \in L_i \text{ (By Definition of Union)} \\ &\iff \exists i \geq 1 : w^R \in (L_i)^R \text{ (By Definition of Reversal)} \\ &\iff w^R \in \bigcup_{i=1}^{\infty} (L_i)^R \text{ (By Definition of Union)}. \end{aligned}$$

Hence $(\bigcup_{i=1}^{\infty} L_i)^R = \bigcup_{i=1}^{\infty} (L_i)^R$. Hence the proof of (b). \square

Lemma 28. a) $\forall x, y \in \Sigma^* : (x \sqcup y)^R = x^R \sqcup y^R$.

b) $\forall x \in \Sigma^* \forall n \geq 0 : (x^{\sqcup, n})^R = (x^R)^{\sqcup, n}$.

c) $\forall x \in \Sigma^* : (x^{\sqcup, *})^R = (x^R)^{\sqcup, *}$.

Proof. a) Let $w \in x \sqcup y$. $w^R \in (x \sqcup y)^R$. **Induction Basis:** It has two cases, when $x = \varepsilon$ and $y = \varepsilon$. **Case 1:** When $x = \varepsilon$

$$\begin{aligned} w^R &\in (\varepsilon \sqcup y)^R \text{ (Since } x = \varepsilon) \\ w^R &\in \{y\}^R \text{ (Since } (\varepsilon \sqcup y) = \{y\}) \\ w^R &\in \{y^R\} \text{ (Since } \{y\}^R = \{y^R\}) \\ w^R &\in \varepsilon^R \sqcup y^R \text{ (Since } \{y^R\} = (\varepsilon \sqcup y^R) \text{ and } \varepsilon = \varepsilon^R). \end{aligned}$$

Case 2: When $y = \varepsilon$ is similar to Case 1. **Induction Step:** Let $x = au$ and $y = bv$. $x^R = (au)^R$ and $y^R = (bv)^R$ that implies $x^R = u^R a^R$ and $y^R = v^R b^R$.

$$\begin{aligned} (x \sqcup y)^R &= (au \sqcup bv)^R \text{ (Since } x = au \text{ and } y = bv) \\ &= (a(u \sqcup bv) \cup b(au \sqcup v))^R \text{ (By Definition 39)} \\ &= (a \cdot (u \sqcup bv))^R \cup (b \cdot (au \sqcup v))^R \text{ (By (a) in Lemma 27)} \\ &= (u \sqcup bv)^R \cdot a^R \cup (au \sqcup v)^R \cdot b^R \text{ (Since } (a \cdot b)^R = b^R \cdot a^R) \\ &= (u^R \sqcup (bv)^R) \cdot a^R \cup ((au)^R \sqcup v)^R \cdot b^R \text{ (By Induction Hypothesis)} \\ &= (u^R \sqcup v^R b^R) \cdot a^R \cup (u^R a^R \sqcup v)^R \cdot b^R \text{ (Since } (a \cdot b)^R = b^R \cdot a^R) \\ &= u^R a^R \sqcup v^R b^R \text{ (By Lemma 26)} \\ &= (au)^R \sqcup (bv)^R \text{ (Since } (a \cdot b)^R = b^R \cdot a^R) \\ &= x^R \sqcup y^R \text{ (Since } x^R = u^R a^R \text{ and } y^R = v^R b^R). \end{aligned}$$

Hence the proof of (a).

b) We prove this by induction on n . Induction Basis: When $n = 0$. Then we have

$$\begin{aligned}
(x^{\sqcup,0})^R &= \{\varepsilon\}^R \text{ (By Definition 40)} \\
&= \{\varepsilon^R\} \text{ (Since } \{\varepsilon\}^R = \{\varepsilon^R\}\text{)} \\
&= \{\varepsilon\} \text{ (Since } \varepsilon = \varepsilon^R\text{)} \\
&= (x^R)^{\sqcup,0} \text{ (By Definition 40).}
\end{aligned}$$

Induction Step:

$$\begin{aligned}
(x^{\sqcup,n})^R &= (x^{\sqcup,n-1} \sqcup x)^R \text{ (By Definition 40)} \\
&= (x^{\sqcup,n-1})^R \sqcup x^R \text{ (By (a) in Lemma 28)} \\
&= (x^R)^{\sqcup,n-1} \sqcup x^R \text{ (By Induction Hypothesis)} \\
&= (x^R)^{\sqcup,n} \text{ (By Definition 40).}
\end{aligned}$$

Hence the proof of (b).

c)

$$\begin{aligned}
(x^{\sqcup,*})^R &= \left(\bigcup_{n=0}^{\infty} x^{\sqcup,n} \right)^R \text{ (By Definition 40)} \\
&= \bigcup_{n=0}^{\infty} (x^{\sqcup,n})^R \text{ (By (b) in Lemma 27)} \\
&= \bigcup_{n=0}^{\infty} (x^R)^{\sqcup,n} \text{ (By (b) in Lemma 28)} \\
&= (x^R)^{\sqcup,*} \text{ (By Definition 40)}
\end{aligned}$$

Hence the proof of (c). □

Lemma 29. a) $\forall L_1 \subseteq \Sigma^* L_2 \subseteq \Sigma^* : (L_1 \sqcup L_2)^R = (L_1)^R \sqcup (L_2)^R$.
b) $\forall L \subseteq \Sigma^* : (L^{\sqcup,*})^R = ((L)^R)^{\sqcup,*}$.

Proof. a) Let $w \in L_1 \sqcup L_2$. $w^R \in (L_1 \sqcup L_2)^R$. Let $x \in L_1, y \in L_2 : w^R \in (x \sqcup y)^R$. Now we claim that $w^R \in x^R \sqcup y^R$ which is true by (a) in Lemma 28. Therefore, $w^R \in (L_1)^R \sqcup (L_2)^R$. Similarly, if $w^R \in (L_1)^R \sqcup (L_2)^R$ then $w^R \in (L_1 \sqcup L_2)^R$. Hence $(L_1 \sqcup L_2)^R = (L_1)^R \sqcup (L_2)^R$. Hence the proof of (a).
b) Let $w \in L^{\sqcup,*}$. $w^R \in (L^{\sqcup,*})^R$. Let $x \in L : w^R \in (x^{\sqcup,*})^R$. Now we claim that $w^R \in (x^R)^{\sqcup,*}$ which is true by (c) in lemma 28. Therefore, $w^R \in ((L)^R)^{\sqcup,*}$. Similarly, if $w^R \in ((L)^R)^{\sqcup,*}$ then $w^R \in (L^{\sqcup,*})^R$. Hence $(L^{\sqcup,*})^R = ((L)^R)^{\sqcup,*}$. □

Theorem 18. *SHUF is closed under mirror image.*

Proof. Before proving this theorem let us define the reversal function as follows: *Reverse* is a function from a SHUF expression to a SHUF expression such that if S is a SHUF expression, then $L(\text{Reverse}(S)) = (L(S))^R$.

We have the following for each SHUF expression: If \emptyset and each $w \in \Sigma^*$ are SHUF expressions, then $\emptyset \mapsto \emptyset$, $w \mapsto w^R$ and if S_1, S_2 are SHUF expressions, then $(S_1 + S_2) \mapsto (\text{Reverse}(S_1) + \text{Reverse}(S_2))$, $(S_1 \sqcup S_2) \mapsto (\text{Reverse}(S_1) \sqcup \text{Reverse}(S_2))$ and $((S_1)^{\sqcup,*}) \mapsto ((\text{Reverse}(S_1))^{\sqcup,*})$.

Claim: $\forall L \in \mathcal{SHUF} \Rightarrow L^R \in \mathcal{SHUF}$. Assume that L is defined by some SHUF expression S . We show that there is another SHUF expression $\text{Reverse}(S)$ such that $L(\text{Reverse}(S)) = (L(S))^R$, i.e., the language of $\text{Reverse}(S)$ is the reversal of the language of S .

We prove this by structural induction. **Induction Basis:** We have two cases, \emptyset and w for $w \in \Sigma^*$. If S is a SHUF expression and S is one of \emptyset or w , then $L(S)$ would be $L(\emptyset) = \emptyset$ or $L(w) = \{w\}$, respectively.

Case 1:

If $S = \emptyset$, then $L(S) = L(\emptyset) = \emptyset = \emptyset^R = L(\text{Reverse}(\emptyset)) = L(\text{Reverse}(S))$. Therefore, $(L(S))^R = L(\text{Reverse}(S))$.

Case 2:

If $S = w$, then $L(S) = L(w) = \{w\} \subseteq \Sigma^*$ and $L(\text{Reverse}(S)) = L(w^R) = \{w^R\} \subseteq \Sigma^*$. Now, $(L(S))^R = \{w\}^R = \{w^R\} = L(\text{Reverse}(S))$. Therefore, $(L(S))^R = L(\text{Reverse}(S))$.

Induction: Let S_1, S_2 are SHUF expressions such that $L(S_1), L(S_2)$ are in \mathcal{SHUF} .

$$\begin{aligned}
(L(S_1 + S_2))^R &= (L(S_1) \cup L(S_2))^R \text{ (By Definition 45)} \\
&= (L(S_1))^R \cup (L(S_2))^R \text{ (By (a) in Lemma 27)} \\
&= L(\text{Reverse}(S_1)) \cup L(\text{Reverse}(S_2)) \text{ (By Induction)} \\
&= L(\text{Reverse}(S_1) + \text{Reverse}(S_2)) \text{ (By Definition 45)} \\
&= L(\text{Reverse}(S_1 + S_2)) \text{ (By the Reverse function)}.
\end{aligned}$$

$$\begin{aligned}
(L(S_1 \sqcup S_2))^R &= (L(S_1) \sqcup L(S_2))^R \text{ (By Definition 45)} \\
&= (L(S_1))^R \sqcup (L(S_2))^R \text{ (By (a) in Lemma 29)} \\
&= L(\text{Reverse}(S_1)) \sqcup L(\text{Reverse}(S_2)) \text{ (By Induction)} \\
&= L(\text{Reverse}(S_1) \sqcup \text{Reverse}(S_2)) \text{ (By Definition 45)} \\
&= L(\text{Reverse}(S_1 \sqcup S_2)) \text{ (By the Reverse function)}.
\end{aligned}$$

$$\begin{aligned}
(L(S_1^{\sqcup,*}))^R &= ((L(S_1))^{\sqcup,*})^R \text{ (By Definition 45)} \\
&= ((L(S_1))^R)^{\sqcup,*} \text{ (By (b) in Lemma 29)} \\
&= (L(\text{Reverse}(S_1)))^{\sqcup,*} \text{ (By Induction)} \\
&= L((\text{Reverse}(S_1))^{\sqcup,*}) \text{ (By Definition 45)}
\end{aligned}$$

□

3.5 Representations and Normal Forms

Our desired representation theorem is stated as follows:

Theorem 19 (Representation Theorem 1). *Let $L \in \mathcal{JFA}$. Then there exists a number $n \geq 1$ and finite sets M_i, N_i for $1 \leq i \leq n$, so that the following representation is valid.*

$$L = \bigcup_{i=1}^n \text{perm}(M_i) \sqcup (\text{perm}(N_i))^{\sqcup,*} \quad (3.1)$$

We will prove this representation theorem on the level of α -SHUF expressions, so that we actually get a normal form theorem for these. A central tool in the proofs of this normal form theorem is the following notion that corresponds to the well-known star-height of regular expressions. Let us recall the following definitions of *star-height* and *string form*; see [15, 16, 48].

Definition 47. *The star-height $h_\alpha(E)$ of a regular expression E is inductively defined as follows:*

- $h_\alpha(\emptyset) = h_\alpha(\varepsilon) = h_\alpha(a) = 0$ for $a \in \Sigma$.
- $h_\alpha(E_1 \cup E_2) = h_\alpha(E_1 E_2) = \max\{h_\alpha(E_1), h_\alpha(E_2)\} \wedge h_\alpha(E^*) = h_\alpha(E) + 1$.

Definition 48. *The star-height $h(R)$ of a regular language R is defined by $h(R) = \min\{h_\alpha(E) : E \text{ is a regular expression denoting } R\}$.*

Thus, for any regular expression E , $h_\alpha(E)$ is the maximum length of a sequence of stars in the expression E , such that each star is in the scope of the star that follows it. $h(E)$, however, indicates the star height of the language $L(E)$ denoted by E as defined above. Obviously, for any regular language R , $h(R) \geq 1$ iff R is infinite.

Definition 49. The string form E_s of a regular expression E is defined inductively as follows:

- If $h_\alpha(E) = 0$ then $E_s = w_1 \cup w_2 \cup \dots \cup w_p$ where $w_j \in \Sigma^*$, $j = 1, \dots, p$, $p \geq 0$.
- If $h_\alpha(E) = k > 0$, then $E_s = F_1 \cup F_2 \cup \dots \cup F_p$, $p > 0$ where each F_i is a string of the form:

$$w_1 H_1^* w_2 H_2^* \dots w_l H_l^* w_{l+1}, \quad l > 0,$$
 where $w_j \in \Sigma^*$, H_j are in string form and $h_\alpha(H_j) \leq k - 1$, $j = 1, \dots, l$.

Notice that the string form is very close to the normal form that we have derived for our shuffle expressions.

Example 13. Let $E = (10^*1)^*$. Then $h_\alpha(E) = 2$ is the star height of E . However, $h(E) = 1$ because $E_1 = \varepsilon \cup 1(0 \cup 11)^*1$ is an expression equivalent to E , i.e., $L(E_1) = L(E)$.

Example 14. Let $E = (0 \cup 10^*1)^*$. Here again $h_\alpha(E) = 2$. Moreover, this language has been shown to be of star height 2 by McNaughton [84], using graph-theoretical methods.

Definition 50. We can inductively associate the (shuffle iteration) height h to any α -SHUF expression S as follows.

- If S is a base case, then $h(S) = 0$.
- If $S = (S_1 + S_2)$ or $S = (S_1 \sqcup S_2)$, then $h(S) = \max\{h(S_1), h(S_2)\}$.
- If $S = S_1^{\sqcup, *}$, then $h(S) = h(S_1) + 1$.

The shuffle iteration height of a $L \in \mathcal{JFA}$ is then the smallest shuffle iteration height of any α -SHUF expression S describing L .

Let us mention the following consequence obtained by combining Theorem 19 with Theorem 13, Lemma 22 and Theorem 12.

Corollary 7. $L \in \mathcal{JFA}$ if and only if there is a regular language R of star height at most one such that $L = \text{perm}(R)$.

Immediately from the Definition 50, we obtain from Proposition 14:

Corollary 8. *A language is finite and perm-closed if and only if it can be described by some α -SHUF expression of shuffle iteration height zero.*

Recall that Eggen's Theorem [18] relates the star height of a regular language to its so-called cycle rank, which formalizes loop-nesting in NFA's. Again, the characterization theorems that we derived allow us to conclude that, in short, for any $L \in \mathcal{JFA}$ there exists some finite machine M of cycle rank at most one such that $L_{\text{JFA}}(M) = L$. Corollary 8 means that, in order to show Theorem 19, it is sufficient (and in a sense stronger) to prove the following normal form theorem for α -SHUF expressions. The proof resembles the one given by Jantzen [58] for a different variant of shuffle expressions, but we keep it here, as it shows several technicalities with these notions.

Theorem 20. *For any α -SHUF expression R , an equivalent α -SHUF expression S with $h(S) = 1$ can be constructed that is the union of n α -SHUF expressions S_1, \dots, S_n such that $S_i = F_i \sqcup G_i^{\sqcup, *}$, where $h(F_i) = h(G_i) = 0$, $1 \leq i \leq n$. Moreover, we can assume that $F_i = \bigcup_{j=1}^{n(i)} u_j$ and $G_i = \bigcup_{j=1}^{m(i)} v_j$, where all u_j and v_j are α -SHUF expressions with \sqcup as their only operators.*

Proof. We show the claim by induction on the height of R . If $h(R) = 0$, then $S = R \sqcup \emptyset^{\sqcup, *}$ is an equivalent expression in the desired normal form. Let $h > 0$. Assume now that the result is true for all α -SHUF expressions of height less than h and consider some α -SHUF expression R with $h(R) = h$. By repeatedly applying the distributive law, we can obtain an equivalent α -SHUF expression R' that is of the following form:

$$R' = \bigcup_{j=1}^m \bigsqcup_{k=1}^{k(j)} S_{j,k},$$

where each expression $S_{j,k}$ contains only shuffle and iterated shuffle operators. In a first step, by applying the commutative law of the shuffle, we can order the $S_{j,k}$ such that, slightly abusing notation, $S_{j,1}, \dots, S_{j,b(j)}$ are base cases, and $S_{j,b(j)+1}, \dots, S_{j,k(j)}$ are of the form $S_{j,i} = (T_{j,i})^{\sqcup, *}$.

To simplify the further discussions, we can assume that none of the base cases $S_{j,1}, \dots, S_{j,b(j)}$ is \emptyset , as this would mean that the language $L(\bigsqcup_{k=1}^{k(j)} S_{j,k})$ is empty, and we can omit this part immediately from the union. In the next step, we form $F'_j := \bigsqcup_{k=1}^{b(j)} S_{j,k}$.

Notice that, by Corollary 8, each F'_j represents a finite perm-closed set. Moreover, we define α -SHUF expressions G'_j of iteration height less than h as follows. If

$b(j) = k(j)$, then $G'_j := \emptyset$. Otherwise, $G'_j := \bigcup_{i=b(j)+1}^{k(j)} T_{j,i}$. By using Rule 4 from Proposition 6, one can see that

$$R'' := \bigcup_{j=1}^m F'_j \sqcup (G'_j)^{\sqcup,*}$$

is equivalent to R' . As all G'_j have iteration height less than h , we can apply the induction hypothesis to them and replace G'_j by equivalent expressions

$$\bigcup_{i=1}^{n(j)} F_{j,i} \sqcup G_{j,i}^{\sqcup,*},$$

where each $F_{j,i}$ and each $G_{j,i}$ are α -SHUF expressions of height zero. Rule 4 now yields the following equivalent expression:

$$R''' := \bigcup_{j=1}^m F'_j \sqcup \bigsqcup_{i=1}^{n(j)} (F_{j,i} \sqcup G_{j,i}^{\sqcup,*})^{\sqcup,*}$$

Now, we can apply Rule 6 to avoid nesting of the iterated shuffle. Hence, the following expression is again equivalent:

$$R^{iv} := \bigcup_{j=1}^m F'_j \sqcup \bigsqcup_{i=1}^{n(j)} (F_{j,i} \sqcup (F_{j,i} \cup G_{j,i})^{\sqcup,*} \cup \{\varepsilon\})$$

Finally, setting $F_{j,I} := F'_j \sqcup \bigsqcup_{i \in I} F'_{j,i}$ and $G_{j,I} := \bigcup_{i \in I} (F_{j,i} \cup G_{j,i})$ for $I \subseteq I(j) := \{1, \dots, n(j)\}$, with $F_{j,\emptyset} = F'_j$ and $G_{j,\emptyset} = \emptyset$, and observing that also these α -SHUF expressions are of height zero, we define

$$S := \bigcup_{j=1}^m \bigcup_{I \subseteq I(j)} F_{j,I} \sqcup G_{j,I}^{\sqcup,*}.$$

By the commutative and distributive laws and by Rule 4, S is equivalent to R^{iv} and satisfies all the properties of the theorem, possibly apart from the last sentence, which can be enforced by exhaustively applying the distributive law. \square

As an example for the above theorem consider an α -SHUF expression $R = (a^{\sqcup,*} \sqcup b)^{\sqcup,*}$. An equivalent α -SHUF expression $S = (b \sqcup (b \cup a)^{\sqcup,*}) \cup \{\varepsilon\}$ is constructed by the union of S_1 and S_2 where $S_1 = b \sqcup (b \cup a)^{\sqcup,*}$ and $S_2 = \{\varepsilon\}$ such that $S_1 = F_1 \sqcup G_1^{\sqcup,*}$ where $F_1 = b$ and $G_1 = b \cup a$. Also $S_2 = F_2 \sqcup G_2^{\sqcup,*}$ where $F_2 = \{\varepsilon\}$ and $G_2 = \emptyset$. Here $h(F_1) = h(F_2) = h(G_1) = h(G_2) = 0$.

Also our assumption in the statement is valid in the example as we can evident from the following that all u_j and v_j are α -SHUF expressions with \sqcup as their only operators: $F_1 = \bigcup_{j=1}^{2(1)} u_j$, $F_1 = u_1$; $u_1 = b$ and $F_2 = \bigcup_{j=1}^{2(2)} u_j$, $F_2 = u_1$; $u_1 = \{\varepsilon\}$. Similarly $G_1 = \bigcup_{j=1}^{m(1)} v_j$, $G_1 = v_1 \cup v_2$; $v_1 = b, v_2 = 2$ and $G_2 = \bigcup_{j=1}^{m(2)} v_j$, $G_2 = v_1$; $v_1 = \emptyset$.

The Representation Theorem can also be derived in yet another method, as in [20], where the connection to the definition of semilinear sets is also drawn, though with a different method and background.

As mentioned earlier, regular expressions over free commutative monoids can be re-interpreted as regular expressions dealing with Parikh vectors. As in [29] we have Theorem 19 on the level of α -SHUF expressions, so that we have a normal form theorem for these expressions. Our proof idea was similar to the one that Jantzen presented in [58]. However, in meantime we understood the connections to Parikh's theorem better, so that we presented a different reasoning in [32].

By the results of Meduna and Zemek, we know that \mathcal{JFA} and \mathcal{REG} are two incomparable families of languages. Above, we derived several characterizations of $\mathcal{JFA} \cap \mathcal{FLN} \subseteq \mathcal{REG}$. Let us first explicitly state a characterization of $\mathcal{JFA} \cap \mathcal{REG}$ that can be easily deduced from our previous results.

Proposition 21. *$L \in \mathcal{JFA} \cap \mathcal{REG}$ if and only if $L \in \mathcal{REG}$ and L is perm-closed.*

We mention this, as the class $\mathcal{JFA} \cap \mathcal{REG}$ can be also characterized as follows according to Ehrenfeucht, Haussler and Rozenberg [19]. Namely, they describe this class of (what they call) commutative regular languages as finite unions of periodic languages. We are not giving a definition of this notion here, but rather state an immediate consequence of their characterization in our terminology.

Corollary 9. *A language L is regular and perm-closed if and only if L is the finite union of periodic languages.*

Proof. If L is regular and perm-closed, then language L is the finite union of periodic languages according to [19, Theorem 6.5]. Conversely, as the finite union of perm-closed languages is perm-closed, we can conclude from [19, Theorem 6.5] that the finite union of periodic languages is regular and perm-closed. \square

Let us finally mention, yet another characterization of $\mathcal{JFA} \cap \mathcal{REG}$ that was derived in [75, Theorem 3] and also by us in [32]. Moreover, a relaxed version of the notion of commutativity (of languages) allows a characterization of \mathcal{REG} , as shown by Reutenauer [97]. We would also like to point to [46], where not

only learnability questions of this class of languages were discussed, but also two further normal form representations of $\mathcal{JFA} \cap \mathcal{REG}$ were mentioned. Further algebraic properties of $\mathcal{JFA} \cap \mathcal{REG}$ were presented by Mateescu [80]. A proper subclass of $\mathcal{JFA} \cap \mathcal{REG}$ (star-free commutative languages) was characterized in [9] with the help of shuffle expressions in a certain normal form.

Chapter 4

Scanning Automata and Grammars

Syntactic considerations of digital images have a tradition of about five decades. They should (somehow) reflect methods applied to picture processing. However, one of the basic methods of scanning pictures in practice have not been thoroughly investigated from a more theoretical point of view: that of using space-filling curves [89, 95, 100, 110, 115]. Here, we start such an investigation with what can be considered as the most simple way of defining space-filling curves: scanning an image line after line.

We introduce finite automata that work this way and call them *boustrophedon finite automata*, or BFA for short, a notion derived from the name of ancient forms of writing that run ‘as the ox turns’. This is an automaton model for processing rectangular-shaped digitized bordered pictures that moves its head one by one at each computation step and changes its direction when the borders are visited.

We also consider *returning finite automata*, or RFA for short, finite automata that scan images line by line and does not alters its direction. We prove that both BFA and RFA describe the same class of pictures.

In contrast to other automata models introduced in the literature, designed for processing pictures, our automata BFA and RFA can read each position (and hence each input symbol) only once. So, in a sense, our automata models correspond, in the string case, to one-way automata, while other models generalize the idea of two-way automata.

In the string case, one-way automata and two-way automata can be shown to be equivalent, by the classical crossing sequence argument or by constructions of Vardi [113], but this is no longer true in the two-dimensional case, as we will show.

In the one-dimensional, i. e., in the string world, one-way finite automata rather naturally correspond to (regular) grammars. It is natural to compare our approach to existing grammar models for the two-dimensional case. We do this in two ways. We show that BFAs are equivalent to regular matrix grammars (RMGs) as introduced in a sequence of papers of Rani Siromoney and her co-authors in the early 1970s. These two-dimensional picture languages have connections to the generation of kolam patterns [107, 116].

Secondly, we compare with isometric regular array grammars (IRAG) [17, 114] that have been introduced as the lowest level of the Chomsky hierarchy of grammars that describe two-dimensional languages. Here, we see the technicality that pictures are (usually) no longer restricted to be of rectangular shape. But, when restricted to the generation of rectangular-shaped pictures, IRAGs can be shown to generalize both BFAs and RMGs.

Our work shows tighter connections between finite automata that works on pictures and array grammars of different types; for overviews on this topic, see [55, 63]. We consider pumping lemmas and interchange lemmas for two-dimensional languages that enable to prove proper hierarchy results. Finally, the constructions showing the interrelations between isometric and non-isometric models turn out to be non-trivial.

4.1 Boustrophedon Finite Automata

We now give the main definition of this section, introducing a new automaton model for picture processing.

Definition 51. A boustrophedon finite automaton, or *BFA* for short, specified as a 7-tuple $M = (Q, \Sigma, R, s, F, \#, \square)$, where Q is a finite set of states, Σ is an input alphabet, $R \subseteq Q \times (\Sigma \cup \{\#\}) \times Q$ is a finite set of rules. A rule $(q, a, p) \in R$ is usually written as $qa \rightarrow p$. The special symbol $\# \notin \Sigma$ indicates the border of the rectangular picture that is processed, $s \in Q$ is the initial state, F is the set of final states.

We are now going to discuss the notions of configurations, valid configurations and an according configuration transition to formalize the work of BFAs, based on snapshots of their work.

Let \square be a new symbol indicating an erased position and let $\Sigma_{\#, \square} := \Sigma \cup \{\#, \square\}$. Then $C_M := Q \times (\Sigma_{\#, \square})_+^+ \times \mathbb{N}$ is the set of configurations of M .

A configuration $(p, A, \mu) \in C_M$ is valid if $1 \leq \mu \leq |A|_r$ and, for every i , $1 \leq i \leq \mu - 1$, the i^{th} row equals $\# \square^{|A|_c - 2} \#$, for every j , $\mu + 1 \leq j \leq |A|_r$, the j^{th} row equals $\#w\#$, $w \in \Sigma^{|A|_c - 2}$, and, for some ν , $0 \leq \nu \leq |A|_c - 2$, $w \in \Sigma^{|A|_c - \nu - 2}$, the μ^{th} row equals $\# \square^\nu w\#$, if μ is odd and $\#w \square^\nu \#$, if μ is even.

Notice that valid configurations model the idea of observable snapshots of the work of the BFA.

- If (p, A, μ) and (q, A', μ) are two valid configurations such that A and A' are identical but for one position (i, j) , where $A'[i, j] = \square$ while $A[i, j] \in \Sigma$, then $(p, A, \mu) \vdash_M (q, A', \mu)$ if $pA[i, j] \rightarrow q \in R$.
- If (p, A, μ) and $(q, A, \mu + 1)$ are two valid configurations, then $(p, A, \mu) \vdash_M (q, A, \mu + 1)$ if the μ^{th} row contains only $\#$ and \square symbols, and if $p\# \rightarrow q \in R$.

The reflexive transitive closure of the relation \vdash_M is denoted by \vdash_M^* .

The BFA M is deterministic, or a B DFA for short, if for all $p \in Q$ and $a \in \Sigma \cup \{\#\}$, there is at most one $q \in Q$ with $pa \rightarrow q \in R$.

The language $L(M)$ accepted by M is then the set of all $m \times n$ pictures W over Σ such that

$$(s, \#_m \odot W \odot \#_m, 1) \vdash_M^* (f, \#_m \odot \square_m^n \odot \#_m, m)$$

for some $f \in F$.

Note that the automaton works on a picture with a first and last column of only $\#$ symbols, but only the part in between these border columns is accepted. In other words, the computation starts with scanning the left uppermost corner of the picture and then working through the picture row-by-row, as the ox turns, i. e., the boustrophedon way, until the last entry of the last row is scanned. The following illustrates how a BFA scans some input picture and also how a picture of a valid configuration looks like; it can be seen that the sequence of \square only indicate how far the input has been processed, see Fig. 4.1.

It should be also clear that the representation on the right-hand side of the previous picture contains all information necessary to describe a configuration apart from the state.



Figure 4.1: How M^- processes an input.

Remark 10. Notice that since rules of the form $p\# \rightarrow q$ need not be present in R , so that in some natural sense the classical regular string languages are a special case of BFA languages.

Example 15. The language of a horizontal line L_- is generated by the BFA $M^- = (Q, \Sigma, R, s, F, \#, \square)$ with $Q = \{s, b, w, b', f\}$, $\Sigma = \{0, 1\}$, $R = \{s0 \rightarrow b, b0 \rightarrow b, b\# \rightarrow b, b\# \rightarrow w, w1 \rightarrow w, w\# \rightarrow b', b'0 \rightarrow f, f0 \rightarrow f, f\# \rightarrow f\}$, $\Sigma = \{0, 1\}$, $F = \{f\}$. M^- accepts $L_- = T(L_\perp)$. Reconsider Fig. 4.1. The displayed sample array is not accepted, because the processing is stuck exactly in the snapshot shown on the right-hand side, which belongs to state w . There is no rule digesting the next input symbol 0 here.

Example 16. The set L_L of tokens L of all sizes and of all proportions, formally represented by

$$L_L = \left\{ \begin{pmatrix} x & (\bullet)^n \\ x & x^n \end{pmatrix}_{x^n} : n \geq 1, m \geq 2 \right\}$$

is accepted by BFA $M = (Q, \Sigma, R, s, F, \#, \square)$, where $Q = \{s, s_1, s_2, s_3, s_4, s_5\}$, $\Sigma = \{x, \bullet\}$, $R = \{sx \rightarrow s_5, s_5\bullet \rightarrow s_1, s_1\bullet \rightarrow s_1, s_1\# \rightarrow s_2, s_1\# \rightarrow s_4, s_2\bullet \rightarrow s_2, s_2x \rightarrow s_3, s_3\# \rightarrow s, s_3\# \rightarrow s_4, s_4x \rightarrow s_4\}$, and $F = \{s_4\}$. A pictorial form of this BFA M is given in Fig. 4.2. We show how a sample token of L is accepted by M in Fig. 4.3.

Note that we always write the state information exactly above the symbol which is being read by the automaton at that step (current position in the picture) of computation, this way displaying the configurations. BFAs with unary alphabets Σ can give interesting examples, as we now show.

Example 17. Consider $M = (\{s_0, s_1, s_2\}, \{a\}, \{s_0a \rightarrow s_1, s_1a \rightarrow s_0, s_0\# \rightarrow s_2, s_2a \rightarrow s_2, s_2\# \rightarrow s_2\}, s_0, \{s_2\}, \#, \square)$. Clearly, $L(M) = \{a \oplus a\}^+ \ominus \{a\}_+^+$ is the set of all pictures over $\{a\}$ that have an even number of columns and at least two rows. Notice that the ‘even property’ of the column number needs to be checked only once (in the first row), as row concatenation enforces this property on the other rows, as well. Clearly, M is deterministic.

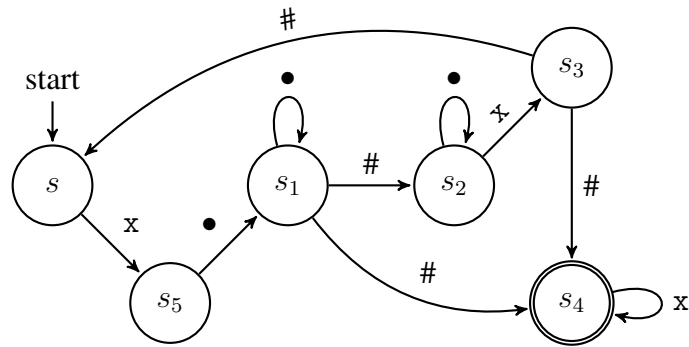


Figure 4.2: BFA M that accepts the language L_L in Example 16

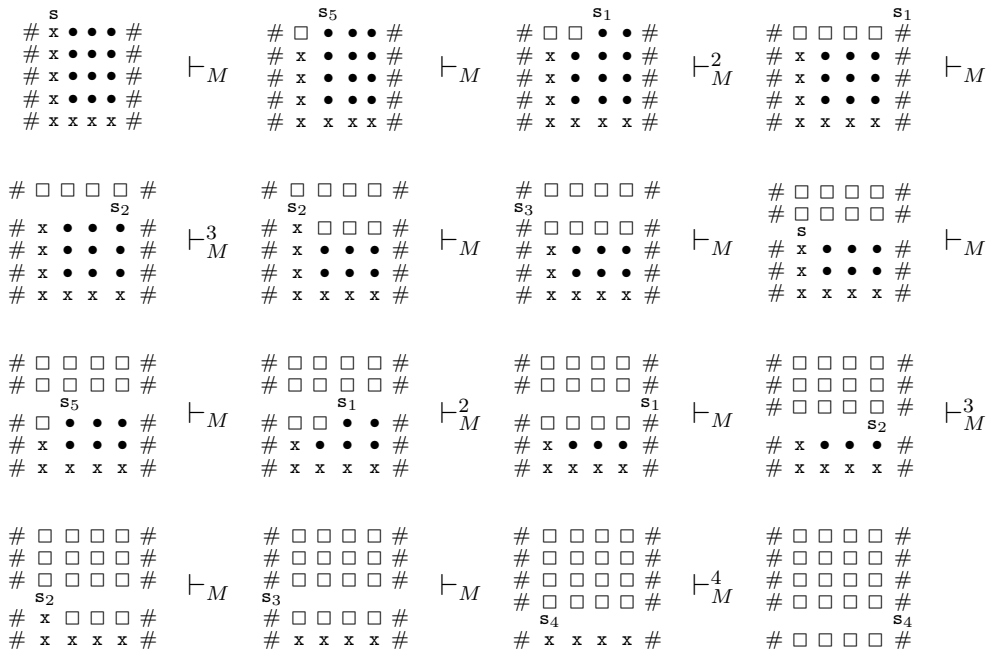


Figure 4.3: Example derivation of the BFA M in Example 16

Here and in the following, if X is some mechanism (automata or grammars) for describing pictures, then $\mathcal{L}_\Sigma(X)$ is the family of non-empty picture languages over the alphabet Σ that can be described by X . If the alphabet does not matter, we can omit the subscript Σ . For instance, $\mathcal{L}_\Sigma(\text{BFA})$ is the family of picture languages $L \subseteq \Sigma^+_{\#}$ such that there is some BFA M with $L = L(M)$.

Remark 11. *We have defined the BFA in such a way that it accepts the picture by reading a last non-# symbol, but we can think of the acceptance by reading a # in the end. In the current BFA model by reading a # it is meant to turn from one row to another. In addition to this we can slightly modify the model to accept the picture by reading the last #, which we wish to remark here as an alternative acceptance for the BFA.*

Now, we derive several normal form characterizations for our picture language class $\mathcal{L}(\text{BFA})$. Later, we will briefly discuss so-called 3-way automata and 4-way automata (see [41]).

Theorem 22. *Let Σ be some alphabet. Then, $\mathcal{L}_\Sigma(\text{BDFA}) = \mathcal{L}_\Sigma(\text{BFA})$.*

Proof. Use the well-known subset construction for determinizing finite automata. This works out, as our BFAs are syntactically the same as classical finite automata, only the interpretation of their processing is different. \square

One of the benefits of having deterministic models is that it usually entails closure under complementation. This is also the case for picture processing automata, as we will later see.

Remark 12. *As we have with word-processing automata, we can insist on having complete BFA and BDFA, meaning that for each state and each input symbol, there is at least one successor state. Clearly, we can easily enforce completeness by introducing a trash state, so that this issue does not need further attention.*

Let us now see a normal form for BFAs that ensures direction-awareness which we have introduced in [34].

Definition 52. *Let $M = (Q, \Sigma, R, s, F, \#, \square)$ be a BFA. M is called direction-aware, or d -BFA for short, if there is a mapping $d : Q \rightarrow \{\underline{r}, \underline{\ell}\}$ such that $d(q) = d(p)$ for any rule $qa \rightarrow p$ with $a \in \Sigma$ and $d(q) \neq d(p)$ for any rule $q\# \rightarrow p$. In addition, $d(s) = \underline{r}$.*

Remark 13. *As we had BDFA for the BFA, it is natural to have deterministic d -BFAs, or d -BDFA for short, for the d -BFAs.*

We now prove the *direction-aware normal form lemma*, in short *DANF lemma*.

Lemma 30. $\mathcal{L}_\Sigma(\text{BFA}) = \mathcal{L}_\Sigma(\text{d-BFA})$.

Proof. $\mathcal{L}_\Sigma(\text{d-BFA}) \subseteq \mathcal{L}_\Sigma(\text{BFA})$ is trivial since d-BFAs are a special case of BFAs. Let us prove the other direction $\mathcal{L}_\Sigma(\text{BFA}) \subseteq \mathcal{L}_\Sigma(\text{d-BFA})$. Given a BFA $M = (Q, \Sigma, R, s, F, \#, \square)$. Let us define a (direction-aware) d-BFA $M_d = (Q_d, \Sigma, R_d, (s, \underline{r}), F_d, \#, \square)$ where $Q_d = Q \times \{\underline{r}, \underline{\ell}\}$ with a mapping $d : Q_d \rightarrow \{\underline{r}, \underline{\ell}\}$, $(q, x) \mapsto x$ for $x \in \{\underline{r}, \underline{\ell}\}$, $(s, \underline{r}) \in Q_d$ is the start state with $(s, \underline{r}) \mapsto \underline{r}$, $F_d \subseteq Q_d$ and $F_d = F \times \{\underline{r}, \underline{\ell}\}$ and R_d is defined as follows:

$$\begin{aligned} R_d &= \{(p, x)a \rightarrow (q, x) : pa \rightarrow q \in R \text{ and } x \in \{\underline{r}, \underline{\ell}\}\} \\ &\cup \{(p, x)\# \rightarrow (q, y) : p\# \rightarrow q \in R \text{ and } x, y \in \{\underline{r}, \underline{\ell}\}, x \neq y\}. \end{aligned}$$

The idea of the construction is that the second component in the state allows us to keep track of the direction, depending upon reading odd- or even-numbered rows. \square

Illustration 2. Let us illustrate the DANF Lemma with the BFA M in Figure 4.2. Let us formally define the equivalent d-BFA $M_d = (Q_d, \Sigma, R_d, (s, \underline{r}), F_d, \#, \square)$, where $Q_d = Q \times \{\underline{r}, \underline{\ell}\} = \{s, \dots, s_5\} \times \{\underline{r}, \underline{\ell}\}$, i. e., $Q_d = \{(s, \underline{r}), (s_1, \underline{r}), (s_2, \underline{r}), (s_3, \underline{r}), (s_4, \underline{r}), (s_5, \underline{r}), (s, \underline{\ell}), (s_1, \underline{\ell}), (s_2, \underline{\ell}), (s_3, \underline{\ell}), (s_4, \underline{\ell}), (s_5, \underline{\ell})\}$, i. e., Q_d has 12 states, with a mapping $d : Q_d \rightarrow \{\underline{r}, \underline{\ell}\}$, $(q, x) \mapsto x$ for $x \in \{\underline{r}, \underline{\ell}\}$, $q \in Q_d$, $(s, \underline{r}) \in Q_d$ is the start state with $(s, \underline{r}) \mapsto \underline{r}$, $F_d \subseteq Q_d$ and $F_d = \{s_4\} \times \{\underline{r}, \underline{\ell}\} = \{(s_4, \underline{r}), (s_4, \underline{\ell})\}$ and R_d using the construction in DANF Lemma is defined as follows:

$$\begin{aligned} R_d &= \{(s, \underline{r})\mathbf{x} \rightarrow (s_5, \underline{r}), (s, \underline{\ell})\mathbf{x} \rightarrow (s_5, \underline{\ell}), \} \\ &\cup \{(p, x)\# \rightarrow (q, y) : p\# \rightarrow q \in R \text{ and } x, y \in \{\underline{r}, \underline{\ell}\}, x \neq y\}. \end{aligned}$$

R_d as described in Figure 4.4 (with only useful states), $F_d = \{(s_4, \underline{r}), (s_4, \underline{\ell})\}$. Here we can note that the final state $(s_4, \underline{\ell})$ will be reached if the pictures have even number of rows and the final state (s_4, \underline{r}) will be reached if the pictures have odd number of rows. A sample token of \mathbb{L} accepted by M_d is

$$\begin{array}{cccc} \mathbf{x} & \bullet & \bullet & \bullet \\ \mathbf{x} & \bullet & \bullet & \bullet \\ \mathbf{x} & \bullet & \bullet & \bullet \\ \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \end{array}$$

Let us now see how this sample is accepted by M_d in Figure 4.4. Please note that we always write the state information exactly above the symbol which is being read by the automaton at that step (current position in the picture) of computation.

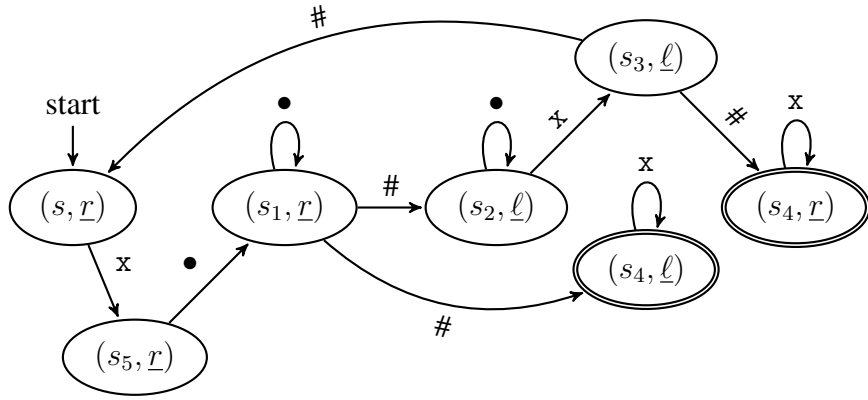
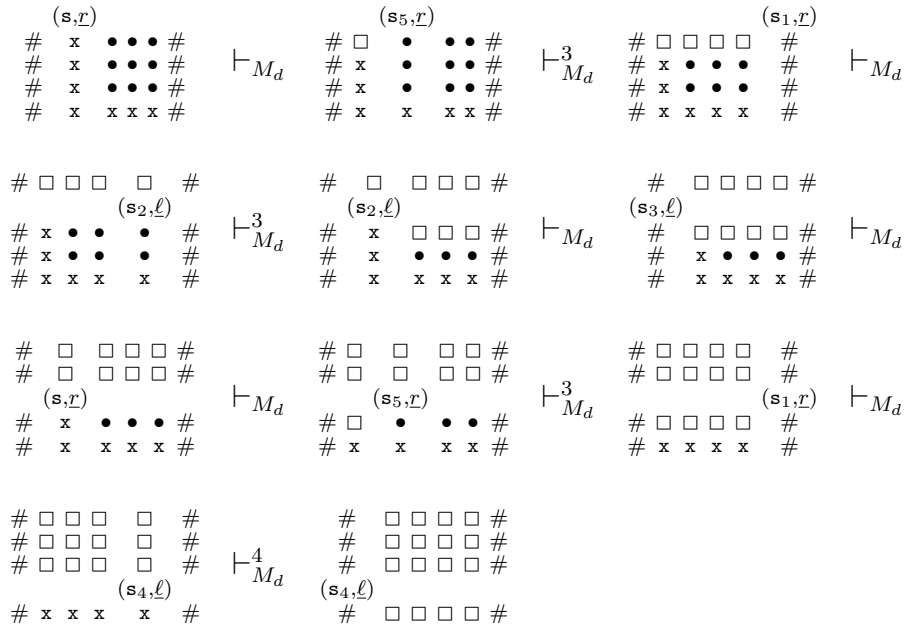


Figure 4.4: d-BFA M_d



As the construction mentioned in DANF Lemma preserves determinism and from Theorem 22 we obtain:

Remark 14. $\mathcal{L}_\Sigma(\text{d-BFA}) = \mathcal{L}_\Sigma(\text{d-BDFA})$.

4.2 Returning Finite Automata

In this section we examine whether the boustrophedon processing mode of our automata is essential. To this end, let us consider yet another interpretation of finite automata, this time termed *returning finite automata*, or RFA for short.

Syntactically, they are identical to BFA, so they can be again described by a 7-tuple $M = (Q, \Sigma, R, s, F, \#, \square)$. However, they always process rows from left to right. Formally, this means that we can carry over all parts of the definition of BFA apart from the notion of a valid configuration, which needs to be slightly modified.

Now, a configuration $(p, A, \mu) \in C_M$ is *valid* if $1 \leq \mu \leq |A|_r$ and, for every i , $1 \leq i \leq \mu - 1$, the i th row equals $\# \square^{|A|_c - 2} \#$, for every j , $\mu + 1 \leq j \leq |A|_r$, the j th row equals $\#w\#$, $w \in \Sigma^{|A|_c - 2}$, and, for some ν , $0 \leq \nu \leq |A|_c - 2$, $w \in \Sigma^{|A|_c - \nu - 2}$, the μ^{th} row equals $\# \square^\nu w\#$.

For instance, M^- from Example 15, viewed as an RFA, will accept L_- again. Now, we return to Example 16, here giving an RFA M' that accepts the language L_L . Consider $M' = (Q, \Sigma, R, s, F, \#, \square)$, where $Q = \{s, s_1, s_2, s_3\}$, $\Sigma = \{x, \bullet\}$, $R = \{sx \rightarrow s_3, s_3\bullet \rightarrow s_1, s_1\bullet \rightarrow s_1, s_1\# \rightarrow s_2, s_1\# \rightarrow s, s_2x \rightarrow s_2\}$, and $F = \{s_2\}$; see Fig. 4.5.

Theorem 23. *Let Σ be some alphabet. Then, $\mathcal{L}_\Sigma(\text{BFA}) = \mathcal{L}_\Sigma(\text{RFA})$.*

Proof. We first show how a RFA can simulate a BFA. The basic idea summarized as follows. On the first row, which is scanned from left to right by both automata, the RFA simulates the BFA one to one. Assume that the BFA, while moving on to the second row, changes into a state q , scans the row from right to left and enters a state p when the beginning of this row is reached.

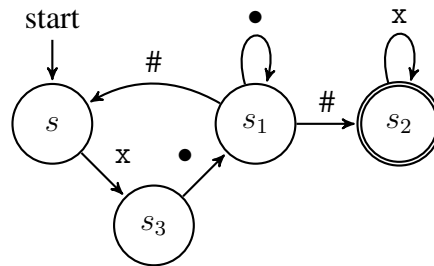


Figure 4.5: RFA M' that accepts the language L_L in Example 16

In order to simulate this behaviour, the RFA stores its current state q in the finite state control and guesses the state p . It then scans the second row from left to right (starting in state p) by applying the transitions of the BFA in the reverse direction. When the end of the row is reached, the computation only proceeds if the RFA is in state q . This procedure is then repeated.

More formally, the states of the RFA like BFA states or triples thereof. These triples simulate the processing of even rows and are like (ℓ, q, r) , where q is the actual state, r is the state that the RFA should reach after finishing the current even row at the right border and ℓ is the state in which the RFA starts simulating the current row (left border). The formal definition is as follows.

Let $M = (Q, \Sigma, R, s, F, \#, \square)$ be some BFA. Then, the equivalent RFA $M' = (Q', \Sigma, R', s, F', \#, \square)$ is defined by $Q' = Q \cup (Q \times Q \times Q)$,

$$\begin{aligned} R' &= \{pa \rightarrow q \mid pa \rightarrow q \in R, a \in \Sigma\} \\ &\cup \{(\ell, q, r)a \rightarrow (\ell, p, r) \mid pa \rightarrow q \in R, \ell, r \in Q, a \in \Sigma\} \\ &\cup \{p\# \rightarrow (\ell, \ell, q) \mid p\# \rightarrow q \in R, \ell \in Q\} \\ &\cup \{(\ell, r, r)\# \rightarrow q \mid \ell\# \rightarrow q \in R, r \in Q\}, \end{aligned}$$

and $F' = F \cup \{(\ell, q, q) \mid \ell \in F, q \in Q\}$. The idea is that the rules from R are now (only) deriving the odd-numbered rows, while for the simulation of the even-numbered rows, we borrow the mirror-image construction well-known from classical formal language theory. Hence, the first component ℓ of some triple $(\ell, q, r) \in Q \times Q \times Q$ memorizes the state associated to the first symbol of that row, q is the current state and r is associated to the last symbol in the row. Reading $\#$ switches between both simulation modes. The formal (induction) proof of the correctness of the construction is carved via the basic observations that can be summarized as follows:

1. Both the given BFA and the simulating RFA work in exactly the same way on rows with an odd number.
2. If $\#a_1 \dots a_n\#$ describes an even-numbered row and the BFA just starts processing it, say, in state r , then it will read it from right to left. After having processed the whole row, the BFA will be in state ℓ . The claim is that then there the simulating RFA will read this row from left to right, starting in state (ℓ, ℓ, r) and terminating in state (ℓ, r, r) after reading the whole row.
3. Conversely, if the simulating RFA starts processing an even-numbered row in state (ℓ, ℓ, r) , finishing it off in state (ℓ, r, r) , then the original BFA can read this row from right to left, starting in state r and terminating in state ℓ .

4. If the whole input array has an odd number of rows, the BFA will read the last row from left to right and the same will be done by the simulating RFA, so that such an array will be accepted by the BFA if and only if it will be accepted by the simulating RFA.
5. If the whole input array has an even number of rows, the BFA will read the last row from right to left, starting in state r and terminating in state ℓ ; clearly, the array is accepted if and only if ℓ is a final state. According to the previous discussions, this is the case if and only if the RFA processes this last starting in state (ℓ, ℓ, r) and finishing in state (ℓ, r, r) , with the additional property that the array is accepted if and only if ℓ is final, so if and only if the original BFA had accepted this array.

The converse direction, simulating RFAs with BFAs, can be seen in a similar way. We give the formal construction as follows:

Let $M = (Q, \Sigma, R, s, F, \#, \square)$ be some RFA. Then, the equivalent BFA $M' = (Q', \Sigma, R', s, F', \#, \square)$ is defined by $Q' = Q \cup (Q \times Q \times Q)$,

$$\begin{aligned}
R' &= \{pa \rightarrow q \mid pa \rightarrow q \in R, a \in \Sigma\} \\
&\cup \{(r, q, \ell)a \rightarrow (r, p, \ell) \mid pa \rightarrow q \in R, \ell, r \in Q, a \in \Sigma\} \\
&\cup \{p\# \rightarrow (r, r, q) \mid p\# \rightarrow q \in R, r \in Q\} \\
&\cup \{(r, \ell, \ell)\# \rightarrow q \mid r\# \rightarrow q \in R, \ell \in Q\},
\end{aligned}$$

and $F' = F \cup \{(r, q, q) \mid r \in F, q \in Q\}$. The idea is similar to the idea of simulating BFAs with RFAs as we had above. We again implement a variant of the mirror-image construction from classical formal language theory on even-numbered rows. Such a row is started in the state (r, r, q) , replacing state q in the given RFA, which means that r is guessed as the target state that the RFA would reach after scanning this row, i. e., after reading the right end of the row. Correspondingly, the transition $(r, q, \ell)a \rightarrow (r, p, \ell)$ not only simulates the RFA transition $pa \rightarrow q$ in a reverse fashion, but also keeps track of the target state r of the simulated RFA (to be reached at the right end of the row), as well as of the state ℓ with which the RFA started its processing of the row (at the left end). Applying $(r, \ell, \ell)\# \rightarrow q$ not only simulates the RFA transition $r\# \rightarrow q$, but also checks if the state ℓ was reached that was memorized as the state that the simulated RFA entered upon starting to read the left end of the row. The formal proof can be done by induction. \square

Correctness of the construction of Theorem 23

In this section observations 2 and 3 are proved in Lemma 31 where we will prove that the BFA and the simulating RFA are same for the current even-numbered row, if this current even-numbered row is assumed to be the last row then observation 5 is also true if $\ell \in F$ and $(\ell, r, r) \in F'$.

For observations 1 and 4 we need more lemmas which shall mainly connect the # rules which makes the switching process between both of the modes, we do not give explicitly here as these lemmas can also proved similar to Lemma 31.

A configuration of the BFAs for the odd rows can be described as follows and it is the same for the RFAs since both are moving from left to right for the odd numbered rows:

$$C_{\text{ODD}}^{\text{BFA}} = \left(\begin{array}{cccccc} \# & \square & \square & \square & \dots & \square & \# \\ \# & \square & \square & \square & \dots & \square & \# \\ \# & \vdots & \vdots & \vdots & \ddots & \vdots & \# \\ \# & \square & \square & a & \dots & b & \# \\ \# & \vdots & \vdots & \vdots & \ddots & \vdots & \# \end{array} \right), q = C_{\text{ODD}}^{\text{RFA}}$$

Even-numbered rows are interesting to see, as the BFAs and RFAs differ in its configuration in this case and those can be described as follows:

$$C_{\text{EVEN}}^{\text{BFA}} = \left(\begin{array}{cccccc} \# & \square & \square & \square & \dots & \square & \# \\ \# & \square & \square & \square & \dots & \square & \# \\ \# & \vdots & \vdots & \vdots & \ddots & \vdots & \# \\ \# & b & \dots & a & \square & \square & \# \\ \# & \vdots & \vdots & \vdots & \ddots & \vdots & \# \end{array} \right), q$$

$$C_{\text{EVEN}}^{\text{RFA}} = \left(\begin{array}{cccccc} \# & \square & \square & \square & \dots & \square & \# \\ \# & \square & \square & \square & \dots & \square & \# \\ \# & \vdots & \vdots & \vdots & \ddots & \vdots & \# \\ \# & \square & \square & a' & \dots & b' & \# \\ \# & \vdots & \vdots & \vdots & \ddots & \vdots & \# \end{array} \right), (\ell, q, r)$$

Having these pictures in mind, now we can consider only the current row that the BFAs (RFAs) are processing and prove the correctness of the construction via the induction on the length of that current row which is the number of the columns n (we make n as fixed).

As mentioned earlier, let us do this for an even-numbered row. Before applying induction let us capture now all the three scenarios (beginning, intermediate and final) in the case of the even-numbered row as follows:

Beginning: BFA: $(\#a_1 \dots a_n \#, r)$ and RFA: $(\#a_1 \dots a_n \#, (\ell, \ell, r))$.

Intermediate: BFA: $(\#a_1 \dots a_m \square \dots \square \#, p) \vdash_{pa_m \rightarrow q} (\#a_1 \dots a_{m-1} \square \dots \square \#, q)$
and RFA: $(\# \square \dots \square a_m a_{m+1} \dots a_n \#, (\ell, q, r)) \vdash_{(\ell, q, r) a_m \rightarrow (\ell, p, r)} (\# \square \dots \square a_{m+1} \dots a_n \#, (\ell, p, r))$.

Final: BFA: $(\# \square \dots \square \#, \ell)$ and RFA: $(\# \square \dots \square \#, (\ell, r, r))$.

So, the general argument can be stated as in the following Lemma 31.

Lemma 31. *Let M and M' be the BFA and RFA as given in the first part of construction of Theorem 23. Let d be mapping as defined in the Definition 52. Then $\forall \ell, r \in Q : d(\ell) = d(r) = \underline{\ell} \forall n \in \mathbb{N} \forall w \in \Sigma^n (\#w\#, (\ell, \ell, r)) \vdash_{M'}^n (\# \square^n \#, \{(\ell, r, r) \in Q' \mid (\#w\#, r) \vdash_M^n (\# \square^n \#, \ell)\})$.*

Proof. We prove this lemma by the induction on n which is the length of the string that is processed in the current even-numbered row and its same as the number of the computations that the BFA (or RFA) would need to process this string.

Induction Basis: When $n = 1$ then $w = a_1$ and $\forall \ell, r \in Q : d(\ell) = d(r) = \underline{\ell} (\#a_1\#, (\ell, \ell, r)) \vdash_{M'}^1 (\# \square^1 \#, \{(\ell, r, r) \in Q' \mid (\#a_1\#, r) \vdash_M^1 (\# \square^1 \#, \ell)\})$.

Induction Hypothesis: $\forall k \in \mathbb{N} 1 \leq k < n \forall \ell, r \in Q : d(\ell) = d(r) = \underline{\ell} (\#w\#, (\ell, \ell, r)) \vdash_{M'}^k (\# \square^k \#, \{(\ell, r, r) \in Q' \mid (\#w\#, r) \vdash_M^k (\# \square^k \#, \ell)\})$.

Induction Step: Let $Q'_w = \{(\ell, r, r) \in Q' \mid (\#w\#, r) \vdash_M^n (\# \square^n \#, \ell)\}$. Q'_w is nothing but the set of states that are reachable by w using the rules from the RFA M' , so we can note that $Q'_w \subseteq Q'$. Now our Claim is that $\forall \ell, r \in Q : d(\ell) = d(r) = \underline{\ell} \forall n \in \mathbb{N} \forall w \in \Sigma^n (\#w\#, (\ell, \ell, r)) \vdash_{M'}^n (\# \square^n \#, Q'_w)$.

So, $\forall \ell, r \in Q : d(\ell) = d(r) = \underline{\ell} \forall n \in \mathbb{N} \forall w \in \Sigma^n$

$$\begin{aligned}
& (\#w\#, (\ell, \ell, r)) \vdash_{M'}^n (\# \square^n \#, Q'_w) \\
& \iff \\
& (\#w\#, (\ell, \ell, r)) (\vdash_{M'}^{n-1} \circ \vdash_{M'}^1) \\
& (\# \square^n \#, \{(\ell, r, r) \in Q' \mid (\#w\#, r) (\vdash_M^{n-1} \circ \vdash_M^1) (\# \square^n \#, \ell)\}) \\
& \iff \\
& \exists Q'' \subseteq Q'_w \exists a_n \in \Sigma \\
& [(\#w\#, (\ell, \ell, r)) \vdash_{M'}^{n-1} (\# \square^{n-1} a_n \#, Q'') \vdash_{M'}^1 (\# \square^n \#, Q'_w)] \\
& \iff \\
& \exists Q'' \subseteq Q'_w \exists a_1, a_n \in \Sigma [\exists x \in \Sigma^{n-2} : w = a_1 \cdot x \cdot a_n \exists p \in Q : d(p) = \underline{\ell} \\
& (\#a_1 \cdot x \cdot a_n \#, (\ell, \ell, r)) \vdash_{M'}^{n-1} \\
& (\# \square^{n-1} a_n \#, \{(\ell, p, r) \in Q' \mid (\#a_1 \cdot x \cdot a_n \#, r) \vdash_{M'}^{n-1} (\#a_1 \square^{n-1} \#, p)\}) \vdash_{M'}^1 \\
& (\# \square^n \#, \{(\ell, r, r) \in Q' \mid (\#a_1 \square^{n-1} \#, p) \vdash_M^1 (\# \square^n \#, \ell)\})] \\
& \square
\end{aligned}$$

By this lemma we are done with a part (observations 2 and 3) of one direction of the correctness proof. As mentioned earlier, the parts of converse direction (simulating RFAs with BFAs) is similar to the one provided here.

Illustration 3. *Let us now illustrate Theorem 23 with our Example 16. For the BFA $M = (Q, \Sigma, R, s, F, \#, \square)$ given in Example 16, let us now formally define the equivalent RFA $M' = (Q', \Sigma, R', s, F', \#, \square)$, where $Q' = \{s, s_1, \dots, s_5\} \cup (\{s, s_1, \dots, s_5\} \times \{s, s_1, \dots, s_5\} \times \{s, s_1, \dots, s_5\})$, that is Q' has 222 states.*

In order to avoid re-computation we do lazy evaluation: so before writing the rules of R' we use the construction of the above theorem to create the 4 sets of rules (2 sets without $\#$ and 2 sets with $\#$) of R' that we need using the 10 rules (6 rules without $\#$ and 4 rules with $\#$) in R of our BFA M in Example 16.

The 1st set of rules in R' has the following 6 sets of rules (out of which only 4 of them will be useful for us), since for an input from Σ other than $\#$ we have 6 rules in our BFA. And each of these 6 sets has only a single rule in it, resulting in 6 singleton sets, so let us write those 6 single rules in one set as follows:

$$\{s\mathbf{x} \rightarrow s_5, s_5\bullet \rightarrow s_1, s_1\bullet \rightarrow s_1, s_4\mathbf{x} \rightarrow s_4, s_2\bullet \rightarrow s_2, s_2\mathbf{x} \rightarrow s_3\}$$

Please note that above for the first rule ($s\mathbf{x} \rightarrow s_5$) we need not think about the case $\bullet \in \Sigma$, since there is no such rule in our BFA from the starting state s for the

input $\bullet \in \Sigma$. Like this we can ignore some of the rules in this RFA construction as we are simulating it only using our BFA rules. The last two rules in the above set are not useful to us to simulate our RFA here.

The 3rd set of rules in R' has 24 rules (6 rules for each rule with $\#$ in our BFA (4 rules)) out of which we only need the following 12 rules (2 sets (each of these set will have 6 rules) of rules) using the following 2 rules $s_1\# \rightarrow s_2$, $s_1\# \rightarrow s_4$ from our BFA. As mentioned earlier we ignore the other two rules $s_3\# \rightarrow s_2$, $s_3\# \rightarrow s_4$ here.

Using the rule $s_1\# \rightarrow s_2 \in R$ we obtain the following set of rules:

$$\begin{aligned} & \{s_1\# \rightarrow (\ell, \ell, s_2) \mid s_1\# \rightarrow s_2 \in R, \ell \in Q\} \\ = & \{s_1\# \rightarrow (s, s, s_2), s_1\# \rightarrow (s_1, s_1, s_2), \\ & s_1\# \rightarrow (s_2, s_2, s_2), s_1\# \rightarrow (s_3, s_3, s_2), \\ & s_1\# \rightarrow (s_4, s_4, s_2), s_1\# \rightarrow (s_5, s_5, s_2)\}. \end{aligned}$$

Using the rule $s_1\# \rightarrow s_4 \in R$ we obtain the following set of rules:

$$\begin{aligned} & \{s_1\# \rightarrow (\ell, \ell, s_4) \mid s_1\# \rightarrow s_4 \in R, \ell \in Q\} \\ = & \{s_1\# \rightarrow (s, s, s_4), s_1\# \rightarrow (s_1, s_1, s_4), \\ & s_1\# \rightarrow (s_2, s_2, s_4), s_1\# \rightarrow (s_3, s_3, s_4), \\ & s_1\# \rightarrow (s_4, s_4, s_4), s_1\# \rightarrow (s_5, s_5, s_4)\}. \end{aligned}$$

Before applying the 2nd set of rules in R' , we select 6 states out of the 12 states that we have in our hand after processing the 3rd set of Rules in such a way that these states can read x , (using our BFA rules, especially the one that takes us to a state after processing a x , as we could see from our BFA s_5, s_3, s_4 are the states that are reached after processing a x) so now the 6 states would be $(s_5, s_5, s_2), (s_3, s_3, s_2), (s_4, s_4, s_2), (s_5, s_5, s_4), (s_3, s_3, s_4), (s_4, s_4, s_4)$.

The 2nd set of rules in R' has 216 rules (36 rules for each rule (6 rules) without $\#$ in our BFA) out of which we only need 6 rules since after applying the 3rd set of rules, we already ignored the states that we do not require for our construction and selected only 6 states to proceed further. Let us apply a rule from the 2nd set of rules in R' to these 6 states as per our construction (6 rules, each rule for each state) as follows:

$$\begin{aligned} & (s_5, s_5, s_2)x \rightarrow (s_5, s, s_2), (s_3, s_3, s_2)x \rightarrow (s_3, s_2, s_2), (s_4, s_4, s_2)x \rightarrow (s_4, s_4, s_2), \\ & (s_5, s_5, s_4)x \rightarrow (s_5, s, s_4), (s_3, s_3, s_4)x \rightarrow (s_3, s_2, s_4), (s_4, s_4, s_4)x \rightarrow (s_4, s_4, s_4). \end{aligned}$$

Now we have in our hand again 6 states, we could see as per our construction we already have one final state among these 6 states, which is (s_4, s_4, s_4) . The RFA that we have constructed will reach this final state for the pictures that have only 2 rows in it.

Before applying again the 2nd set of rules in R' , we select 2 states out of the 6 states that we have in our hand in such a way that these states can read a \bullet , (using our BFA rules (especially that takes us to a state after processing a \bullet , as we could see from our BFA s_2 is the only state that is reached after processing a \bullet and not the other two states s, s_4) so now the 2 states would be $(s_3, s_2, s_2), (s_3, s_2, s_4)$.

Now we apply again the 2nd set of rules in R' , that is from the 216 rules, we are forced to apply only the 2 rules since we have ignored the states that we do not require for our construction and selected only 2 states to proceed further. Let us apply a rule from the 2nd set of rules in R' to these 2 states as per our construction (2 rules, each rule for each state) as follows:

$$(s_3, s_2, s_2)\bullet \rightarrow (s_3, s_2, s_2), (s_3, s_2, s_4)\bullet \rightarrow (s_3, s_2, s_4).$$

Now we have in our hand again 2 states (actually the same 2 states), and we could see that as per our construction none of them are final states.

The 4th set of rule in R' has 24 rules (6 rules for each rule with $\#$ in our BFA (4 rules)) out of which we only need the following 12 rules (2 sets (each of these set will have 6 rules) of rules) using the following 2 rules ($s_3\# \rightarrow s, s_3\# \rightarrow s_4$) from our BFA. As mentioned earlier we ignore the other two rules $s_1\# \rightarrow s_2, s_1\# \rightarrow s_4$ here.

Using the rule $s_3\# \rightarrow s \in R$ we obtain the following set of rules:

$$\begin{aligned} & \{(s_3, r, r)\# \rightarrow s \mid s_3\# \rightarrow s \in R, r \in Q\} \\ = & \{(s_3, s, s)\# \rightarrow s, (s_3, s_1, s_1)\# \rightarrow s, \\ & (s_3, s_2, s_2)\# \rightarrow s, (s_3, s_3, s_3)\# \rightarrow s, \\ & (s_3, s_4, s_4)\# \rightarrow s, (s_3, s_5, s_5)\# \rightarrow s\}. \end{aligned}$$

Using the rule $s_3\# \rightarrow s_4 \in R$ we obtain the following set of rules:

$$\begin{aligned} & \{(s_3, r, r)\# \rightarrow s_4 \mid s_3\# \rightarrow s_4 \in R, r \in Q\} \\ = & \{(s_3, s, s)\# \rightarrow s_4, (s_3, s_1, s_1)\# \rightarrow s_4, \\ & (s_3, s_2, s_2)\# \rightarrow s_4, (s_3, s_3, s_3)\# \rightarrow s_4, \\ & (s_3, s_4, s_4)\# \rightarrow s_4, (s_3, s_5, s_5)\# \rightarrow s_4\}. \end{aligned}$$

Before applying the 1st set of rules in R' , we select 2 states out of the 12 states that we have now in our hand after processing the 4th set of rules, in such a way that these states can read a x , (using our BFA rules (especially the one that takes us to a state after processing a x , as we could see from our BFA s, s_4 are the states that are reached after processing a x) so now the 2 states would be s, s_4 .

The 1st set of rules in R' has 6 rules (for each rule without $\#$ in our BFA) out of which we only need 2 rules since after applying the 4th set of rules, we already ignored the states that we do not require for our construction and forced to have only 2 states to proceed further. Let us apply a rule from the 1st set of rules in R' to these 2 states as per our construction (2 rules, each rule for each state) as follows: $sx \rightarrow s_5, s_4x \rightarrow s_4$.

Now we have in our hand again 2 states, and we could see that as per our construction we already have one final state among these 2 states, which is s_4 . The RFA that we have constructed will reach this final state for the pictures that have only odd number of rows in it.

The other state s_5 is not a final state, so it will be processed further as we mentioned in the beginning, for the 1st set of rules in R' and will be continued further until it leads to either one of the 2 final states (s_4, s_4, s_4) and s_4 . We can observe that this step is the one that leads to accept pictures with even number of rows other than picture with only 2 number of rows. R' contains 270 rules. These are made explicit now:

$$\begin{aligned}
R' = & \{sx \rightarrow s_5, s_5\bullet \rightarrow s_1, s_1\bullet \rightarrow s_1, s_4x \rightarrow s_4, s_2\bullet \rightarrow s_2, s_2x \rightarrow s_3\} \\
& \cup \{(l, s_5, r)x \rightarrow (l, s, r) \mid l, r \in Q\} \cup \{(l, s_1, r)\bullet \rightarrow (l, s_1, r) \mid l, r \in Q\} \\
& \cup \{(l, s_4, r)x \rightarrow (l, s_4, r) \mid l, r \in Q\} \cup \{(l, s_2, r)\bullet \rightarrow (l, s_2, r) \mid l, r \in Q\} \\
& \cup \{(l, s_3, r)x \rightarrow (l, s_2, r) \mid l, r \in Q\} \cup \{(l, s_1, r)\bullet \rightarrow (l, s_5, r) \mid l, r \in Q\} \\
& \cup \{s_1\# \rightarrow (l, l, s_2) \mid l \in Q\} \cup \{s_1\# \rightarrow (l, l, s_4) \mid l \in Q\} \\
& \cup \{s_3\# \rightarrow (l, l, s) \mid l \in Q\} \cup \{s_3\# \rightarrow (l, l, s_4) \mid l \in Q\} \\
& \cup \{(s_1, r, r)\# \rightarrow s_2 \mid r \in Q\} \cup \{(s_1, r, r)\# \rightarrow s_4 \mid r \in Q\} \\
& \cup \{(s_3, r, r)\# \rightarrow s \mid r \in Q\} \cup \{(s_3, r, r)\# \rightarrow s_4 \mid r \in Q\}, \text{ and}
\end{aligned}$$

$$\text{also } F' = \{s_4, (s_4, s, s), (s_4, s_1, s_1), (s_4, s_2, s_2), (s_4, s_3, s_3), (s_4, s_4, s_4), (s_4, s_5, s_5)\}.$$

But the accepting computation of M' need not go through all of the 270 rules in R' . Instead of drawing such a big automaton, let us draw this RFA M' ignoring the states and rules that we do not require to accept the language in Example 16, so, omitting useless states and transitions involving them (see Fig. 4.6).

This automaton is equivalent to the one given in Fig. 4.5 that has only 4 states which is via the classical state elimination method [64]. Notice again that both the BFA we started with and the RFA that we have obtained are incomplete automata.

Illustration 4. Let us now illustrate the reverse inclusion of Theorem 23 with the RFA in Fig 4.5. For the RFA $M' = (Q, \Sigma, R, s, F, \#, \square)$, with $Q = \{s, s_1, s_2, s_3\}$, $\Sigma = \{x, \bullet\}$, $R = \{sx \rightarrow s_3, s_3\bullet \rightarrow s_1, s_1\bullet \rightarrow s_1, s_1\# \rightarrow s_2, s_1\# \rightarrow s, s_2x \rightarrow s_2\}$, and $F = \{s_2\}$ that accepts the language in Example 16.

Let us now formally define the equivalent BFA $M = (Q', \Sigma, R', s', F', \#, \square)$, where $Q' = Q \cup (Q \times Q \times Q)$. i. e., $Q' = \{s, s_1, s_2, s_3\} \cup (\{s, s_1, s_2, s_3\} \times \{s, s_1, s_2, s_3\} \times \{s, s_1, s_2, s_3\})$, i. e., Q' has 68 states, R' contains 84 rules that are made explicit now:

$$\begin{aligned} R' = & \{sx \rightarrow s_3, s_3\bullet \rightarrow s_1, s_1\bullet \rightarrow s_1, s_2x \rightarrow s_2\} \\ & \cup \{(r, s_3, \ell)x \rightarrow (r, s, \ell) \mid \ell, r \in Q\} \cup \{(r, s_1, \ell)\bullet \rightarrow (r, s_3, \ell) \mid \ell, r \in Q\} \\ & \cup \{(r, s_2, \ell)x \rightarrow (r, s_2, \ell) \mid \ell, r \in Q\} \cup \{(r, s_1, \ell)\bullet \rightarrow (r, s_1, \ell) \mid \ell, r \in Q\} \\ & \cup \{s_1\# \rightarrow (r, r, s) \mid r \in Q\} \cup \{s_1\# \rightarrow (r, r, s_2) \mid r \in Q\} \\ & \cup \{(s_1, \ell, \ell)\# \rightarrow s \mid \ell \in Q\} \cup \{(s_1, \ell, \ell)\# \rightarrow s_2 \mid \ell \in Q\}, \end{aligned}$$

and $F' = \{s_2, (s_2, s, s), (s_2, s_1, s_1), (s_2, s_2, s_2), (s_2, s_3, s_3)\}$.

Let us draw this BFA M by having only useful states (see Fig. 4.7). This automaton M is equivalent to the one in Fig. 4.2 which is again via the classical state elimination method [64]. Here again BFA and RFA are incomplete automata.

Remark 15. We could note that the d -BFA M_d in Fig. 4.4 that was constructed using DANF Lemma is equivalent to the RFA in Fig. 4.6 that was constructed using Theorem 23.

Remark 16. We can also introduce deterministic RFAs, or RDFAs for short. Again with the subset construction, they are easily seen to describe the same family of array languages. Notice that this does not automatically follow from combining Theorem 22 and Theorem 23, as the construction in the latter result will introduce nondeterminism in general.

Finally, notice that for pictures over a unary alphabet, the power of RFAs and BFAs trivially coincide; reconsider the automaton given in Example 17 working as a RFA.

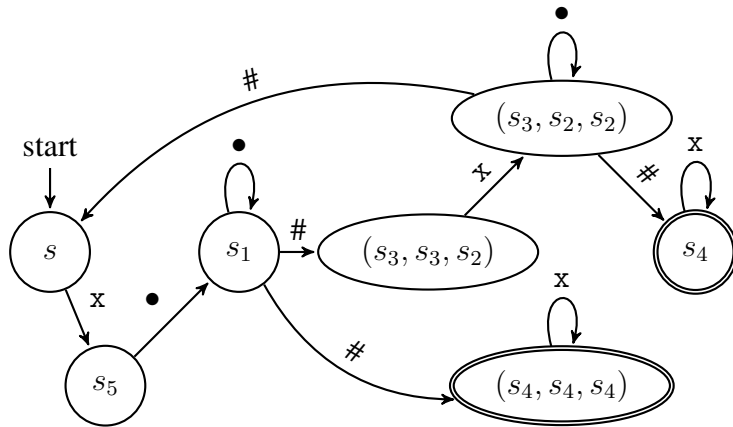


Figure 4.6: RFA M' constructed by Theorem 23 with only useful states

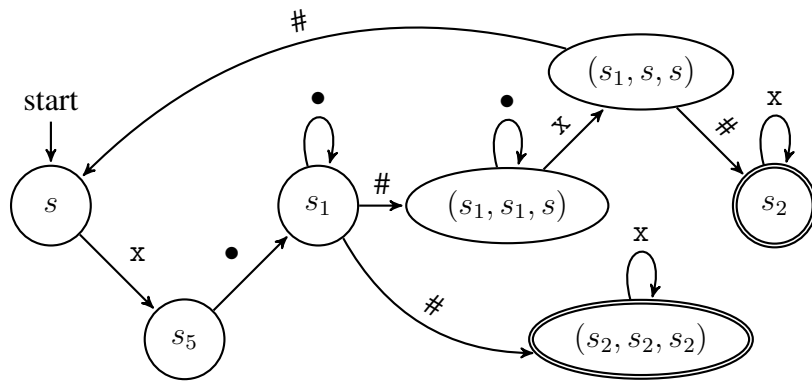


Figure 4.7: BFA M constructed by Theorem 23 with only useful states

4.3 Regular Matrix Languages

We now recall the notion of two-dimensional right-linear grammars (2RLG) as given in [41]. The original definition of a 2RLG (under the name of a regular matrix grammar (RMG)) and the properties of the corresponding class of picture languages, traditionally called RML, standing for *regular matrix languages*, can be found in [71, 105, 106, 111].

Definition 53. A two-dimensional right-linear grammar (RMG for short) is a 7-tuple $G = (V_h, V_v, \Sigma_I, \Sigma, S, R^h, R^v)$, where:

- V_h is a finite set of horizontal non-terminals;
- V_v is a finite set of vertical non-terminals, with $V_h \cap V_v = \emptyset$;
- $\Sigma_I \subseteq V_v$ is a finite set of intermediates;
- Σ is a finite set of terminals;
- $S \in V_h$ is a starting symbol;
- R^h is a finite set of horizontal rules of the form $V \rightarrow AV'$ or $V \rightarrow A$, where $V, V' \in V_h$ and $A \in \Sigma_I$;
- R^v is a finite set of vertical rules of the form $W \rightarrow aW'$ or $W \rightarrow a$, where $W, W' \in V_v$ and $a \in \Sigma$.

In fundamental contrast to the processing of arrays by BFA or by RFA, there are two phases of derivation of a RMG. In the first phase, a horizontal string of intermediate symbols is generated by means of the string grammar $G_h = (V_h, \Sigma_I, S, R^h)$, denoted by $H(G)$.

Note that the length of the intermediate strings generated by $H(G)$ determines the number of columns of the picture that is going to be generated. In the second phase, treating each intermediate as a start symbol, the vertical generation of the actual picture is done in parallel, by applying a finite set of right-linear rules R^v . In order to produce a rectangular-shaped picture, the rules of R^v must be applied in parallel; this also means that the rules of the form $V_i \rightarrow a_i$ are all applied in every column simultaneously to finish the picture with generating its last row. These rules make sure that the columns can grow only in downward direction. For simplicity, we write a single derivation step of $H(G)$ as well a parallel derivation step (in the second phase) as \Rightarrow . Following our conventions, we will denote the corresponding language family by $\mathcal{L}(RMG)$.

We note that our model is closely connected with RMG, as we will show more precisely in the following. The formalization of RMG that we chose is closer to our model than the original one due to Siromoney and her co-authors. Alternatively, we could have referred to finite-state matrix automata as defined in [105]; their work pretty much resembles that of RFAs, but in a sense, they are just the natural automaton model corresponding to the RMG defined above, so that the technicalities that point to the differences between finite-state matrix automata and RFAs are very similar to what we are going to expose below. Let us clarify the working of RMG with two examples.

Example 18. Let us now formally define a RMG G that generates a ‘turned variant’ of the language given in Example 16 as follows. Consider the RMG $G = (V_h, V_v, \Sigma_I, \Sigma, S, R^h, R^v)$, where:

- $V_h = \{S, A\}$;
- $V_v = \{S_1, B, S_2, C\}$;
- $\Sigma_I = \{S_1, S_2\} \subseteq V_v$;
- $\Sigma = \{\bullet, x, y\}$;
- $S \in V_h$ is the starting symbol;
- $R^h = \{S \rightarrow S_1A, A \rightarrow S_1A, A \rightarrow S_2\}$;
- $R^v = \{S_1 \rightarrow \bullet B, B \rightarrow \bullet B, B \rightarrow y, S_2 \rightarrow xC, C \rightarrow xC, C \rightarrow y\}$.

We could see that the language generated by this RMG G , $L(G)$, is the set L'_L of tokens \dashv of all sizes and of all proportions formally represented by

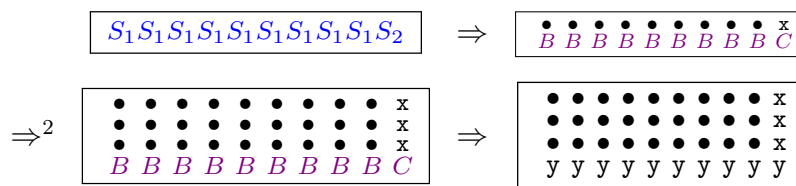
$$L'_L = \left\{ \left(\frac{(\bullet)^n x}{y^n} \right)^{m-1} : n \geq 1, m \geq 2 \right\}.$$

A sample token of \dashv generated by G is

$$\begin{array}{cccccccc} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & x \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & x \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & x \\ y & y & y & y & y & y & y & y & y \end{array}$$

Let us now show how this sample is generated by G ,

$S \Rightarrow S_1A \Rightarrow^8 S_1S_1S_1S_1S_1S_1S_1S_1A \Rightarrow \boxed{S_1S_1S_1S_1S_1S_1S_1S_1S_2}$ describes the work of $H(G)$. From this intermediate string, we can generate the sample in four parallel steps.



Example 19. Consider the RMG $G = (\{S, S_1, S_2\}, \{B, W\}, \{B, W\}, \{0, 1\}, S, \{S \rightarrow BS_1, S_1 \rightarrow BS_1, S_1 \rightarrow WS_2, S_2 \rightarrow BS_2, S_2 \rightarrow B\}, \{W \rightarrow 1W, W \rightarrow 1, B \rightarrow 0B, B \rightarrow 0\})$. As the reader can verify, G describes the language L_{\perp} with one vertical line, which is a ‘turned variant’ of the language L_{-} from Example 15, also see Example 2.

Example 20. Let us reconsider the language of Example 17. This language can be generated by the RMG $G = (\{S_0, S_1\}, \{A, B\}, \{A, B\}, \{a\}, S_0, \{S_0 \rightarrow AS_1, S_1 \rightarrow AS_0, S_1 \rightarrow A\}, \{A \rightarrow aB, B \rightarrow aB, B \rightarrow a\})$.

As we will later see, it is no coincidence that this unary BFA language is also a unary RML. However, in general, we find the following relationship.

Theorem 24. Let Σ be some alphabet. Then, $\mathcal{L}_{\Sigma}(\text{BFA}) = T(\mathcal{L}_{\Sigma}(\text{RMG}))$.

Proof. We provide two simulations to show the claim. Let $G = (V_h, V_v, \Sigma_I, \Sigma, S, R^h, R^v)$ be a RMG. We are going to construct an RFA $M = (Q, \Sigma, R, s, F, \#, \square)$ accepting $T(L(G))$ which is sufficient for the claim $\mathcal{L}_{\Sigma}(\text{BFA}) \supseteq T(\mathcal{L}_{\Sigma}(\text{RMG}))$ thanks to Theorem 23 and Remark 2. Let $Q = (V_h \cup \{f\}) \times (V_v \cup \{f\}) \cup \{s\}$, where $f \notin V_h \cup V_v$, and $F = \{(f, f)\}$. Let R contain the following rules:

- $sa \rightarrow (S', A')$, if $S \rightarrow AS' \in R^h$ and $A \rightarrow aA' \in R^v$,
- $sa \rightarrow (f, A')$, if $S \rightarrow A \in R^h$ and $A \rightarrow aA' \in R^v$,
- $(X, A)a \rightarrow (X, A')$, if $X \in V_h \cup \{f\}$ and $A \rightarrow aA' \in R^v$,
- $(X, A)a \rightarrow (X, f)$, if $A \rightarrow a \in R^v$, $X \in V_h$,
- $(X, f)\# \rightarrow (X', A)$, if $X \rightarrow AX' \in R^h$,
- $(X, f)\# \rightarrow (f, A)$, if $X \rightarrow A \in R^h$,
- $(f, A)a \rightarrow (f, f)$, if $A \rightarrow a \in R^v$.

The idea of the construction is that the generation of columns of G is performed in the second component of the state pairs, whereas the first component corresponds to the generation of the axiom (i. e., the first row of the pictures generated by G). The crucial difference is that the first symbol of the axiom (which in the case of RFA is the first column instead of the first row) is generated and then the first row is generated before the second letter of the axiom is generated in the next row. Hence, the two phases of the picture construction of G are fitting together.

The converse is seen as follows. Let $M = (Q, \Sigma, R, s, F, \#, \square)$ be some RFA. We construct a RMG $G = (V_h, V_v, \Sigma_I, \Sigma, S, R^h, R^v)$ (generating the transposed pictures of $L(M)$) with $V_h = Q \cup \{S\}$, $V_v = \Sigma_I = Q \times Q$, and rules

- $S \rightarrow (s, r)r \in R^h$ for all $r \in Q$,
- $q \rightarrow (q, r)r \in R^h$ for all $q, r \in Q$,
- $q \rightarrow (q, f) \in R^h$ for all $f \in F, q \in Q$,
- $(p, r) \rightarrow a(q, r) \in R^v$ for all $pa \rightarrow q \in R, r \in Q$,
- $(p, r) \rightarrow a \in R^v$ for all $pa \rightarrow q \in R$ and $q\# \rightarrow r \in R$,
- $(p, f) \rightarrow a \in R^v$ for all $pa \rightarrow f \in R, f \in F$.

This concludes the formal construction whose correctness proof can be given by induction. We give the correctness proof idea for the first direction in Theorem 24. Assume that the RMG has generated a single column only then this would be the single row accepted by the RFA, in general if RMG has generated n columns then simulating this RMG can be done by a RFA as per our construction, this general argument can be achieved by induction on n through looking at the sentential form and configuration of considered RMG and RFA respectively at any time of the induction. The correctness of the converse direction can also be viewed in a similar manner. \square

Illustration 5. Let us now illustrate Theorem 24 with the help of Example 18. We first apply the transpose to this language L'_L , which gives $T(L'_L)$, i. e.,

$$T(L'_L) = \left\{ \left(\begin{array}{c} (\bullet)^n \\ \mathbf{x}^n \end{array} \right) \begin{array}{c} \mathbf{y} \\ \mathbf{y} \end{array} \right\} : n \geq 1, m \geq 2 \}.$$

As per the construction given by us in Theorem 24, let us formally define a RFA $M = (Q, \Sigma, R, s, F, \#, \square)$ with $L(M) = T(L'_L)$. Namely, set $Q = \{s, (S, S_1), (S, B), (S, S_2), (S, C), (S, f), (A, S_1), (A, B), (A, S_2), (A, C), (A, f), (f, S_1), (f, B), (f, S_2), (f, C), (f, f)\}$, i. e., Q has 16 states, $F = \{(f, f)\}$ and R has 22 rules which are made explicit now:

- $s\bullet \rightarrow (A, B)$,
- $(S, S_1)\bullet \rightarrow (S, B), (A, S_1)\bullet \rightarrow (A, B), (f, S_1)\bullet \rightarrow (f, B),$
 $(S, B)\bullet \rightarrow (S, B), (A, B)\bullet \rightarrow (A, B), (f, B)\bullet \rightarrow (f, B),$
 $(S, S_2)\mathbf{x} \rightarrow (S, C), (A, S_2)\mathbf{x} \rightarrow (A, C), (f, S_2)\mathbf{x} \rightarrow (f, C),$
 $(S, C)\mathbf{x} \rightarrow (S, C), (A, C)\mathbf{x} \rightarrow (A, C), (f, C)\mathbf{x} \rightarrow (f, C),$
- $(S, B)\mathbf{y} \rightarrow (S, f), (A, B)\mathbf{y} \rightarrow (A, f),$
 $(S, C)\mathbf{y} \rightarrow (S, f), (A, C)\mathbf{y} \rightarrow (A, f),$
- $(S, f)\# \rightarrow (A, S_1), (A, f)\# \rightarrow (A, S_1),$

- $(A, f)\# \rightarrow (f, S_2)$,
- $(f, B)y \rightarrow (f, f)$, $(f, C)y \rightarrow (f, f)$.

The accepting computation of M , need not go through all of the 22 rules, hence by omitting the non-reachable states we obtain RFA M in Fig. 4.8, here we can also note that all the reachable states are useful states and none of the reachable states are useless.

Illustration 6. Let us now illustrate the reverse direction of Theorem 24 with our Example 16 via the RFA in Fig. 4.5. Let $M = (Q, \Sigma, R, s, F, \#, \square)$ be the RFA in Fig. 4.5, so that $L(M) = L_L$. As per the construction given in Theorem 24, we construct the RMG $G = (V_h, V_v, \Sigma_I, \Sigma, S, R^h, R^v)$ with

- $V_h = \{s, s_1, s_2, s_3\} \cup \{S\}$,
- $V_v = \Sigma_I = \{s, s_1, s_2, s_3\} \times \{s, s_1, s_2, s_3\}$, i. e.,
 $\Sigma_I = \{(s, s), (s, s_1), (s, s_2), (s, s_3), (s_1, s), (s_1, s_1), (s_1, s_2), (s_1, s_3),$
 $(s_2, s), (s_2, s_1), (s_2, s_2), (s_2, s_3), (s_3, s), (s_3, s_1), (s_3, s_2), (s_3, s_3)\}$,
- $\Sigma = \{x, \bullet\}$;
- $S \in V_h$ is a starting symbol; and
- $R^h = \{S \rightarrow (s, s)s, S \rightarrow (s, s_1)s_1, S \rightarrow (s, s_2)s_2, S \rightarrow (s, s_3)s_3,$
 $s \rightarrow (s, s)s, s \rightarrow (s, s_1)s_1, s \rightarrow (s, s_2)s_2, s \rightarrow (s, s_3)s_3,$
 $s_1 \rightarrow (s_1, s)s, s_1 \rightarrow (s_1, s_1)s_1, s_1 \rightarrow (s_1, s_2)s_2, s_1 \rightarrow (s_1, s_3)s_3,$
 $s_2 \rightarrow (s_2, s)s, s_2 \rightarrow (s_2, s_1)s_1, s_2 \rightarrow (s_2, s_2)s_2, s_2 \rightarrow (s_2, s_3)s_3,$
 $s_3 \rightarrow (s_3, s)s, s_3 \rightarrow (s_3, s_1)s_1, s_3 \rightarrow (s_3, s_2)s_2, s_3 \rightarrow (s_3, s_3)s_3,$
 $s \rightarrow (s, s_2), s_1 \rightarrow (s_1, s_2), s_2 \rightarrow (s_2, s_2), s_3 \rightarrow (s_3, s_2)\}$;
- $R^v = \{(s, s) \rightarrow x(s_3, s), (s, s_1) \rightarrow x(s_3, s_1), (s, s_2) \rightarrow x(s_3, s_2),$
 $(s, s_3) \rightarrow x(s_3, s_3), (s_3, s) \rightarrow \bullet(s_1, s), (s_3, s_1) \rightarrow \bullet(s_1, s_1),$
 $(s_3, s_2) \rightarrow \bullet(s_1, s_2), (s_3, s_3) \rightarrow \bullet(s_1, s_3), (s_1, s) \rightarrow \bullet(s_1, s),$
 $(s_1, s_1) \rightarrow \bullet(s_1, s_1), (s_1, s_2) \rightarrow \bullet(s_1, s_2), (s_1, s_3) \rightarrow \bullet(s_1, s_3),$
 $(s_2, s) \rightarrow x(s_2, s), (s_2, s_1) \rightarrow x(s_2, s_1), (s_2, s_2) \rightarrow x(s_2, s_2),$
 $(s_3, s_3) \rightarrow x(s_3, s_3), (s_1, s) \rightarrow \bullet, (s_1, s_2) \rightarrow \bullet, (s_3, s) \rightarrow \bullet,$
 $(s_3, s_2) \rightarrow \bullet, (s_2, s_2) \rightarrow x\}$.

$L(G)$ is the set L_L'' of tokens \top of all sizes and of all proportions, formally represented by

$$L_L'' = \left\{ \left(\begin{array}{c} x^n \\ (\bullet \ x)_{m-1} \end{array} \right) : n \geq 1, m \geq 2 \right\}.$$

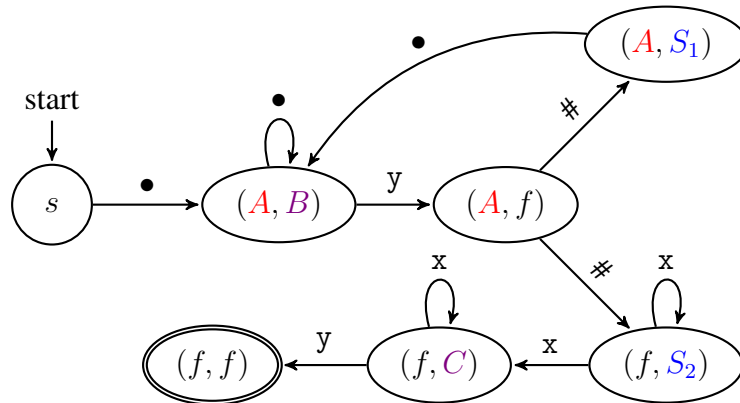


Figure 4.8: RFA M constructed by Theorem 24 with all reachable states

A sample token of \mathbb{T} generated by G is

$x \ x \ x$
 $\bullet \ \bullet \ x$
 $\bullet \ \bullet \ x$
 $\bullet \ \bullet \ x$
 $\bullet \ \bullet \ x$

This sample is generated by G as follows. First, $H(G)$ acts as displayed below; the subsequent parallel derivation steps of G are shown in Fig. 4.9.

$$\begin{aligned}
 S &\Rightarrow (s, s)s \\
 &\Rightarrow (s, s)(s, s_2)s_2 \\
 &\Rightarrow (s, s)(s, s_2)(s_2, s_2)
 \end{aligned}$$

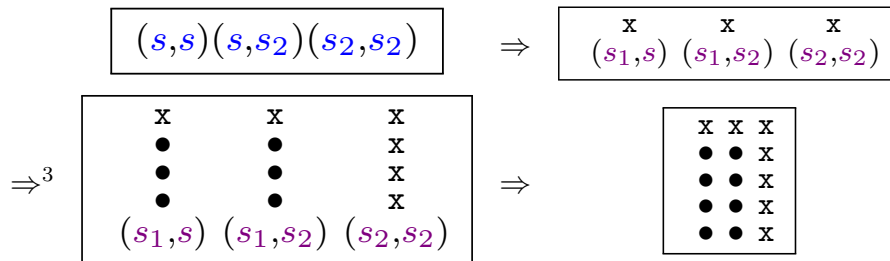


Figure 4.9: A sample parallel derivation of the constructed RMG.

4.4 Regular Array Grammars

Notice that according to our definition, arrays can only have a rectangular shape. These are also known as *non-isometric arrays* in the literature, to distinguish them from *isometric arrays* that are not restricted to rectangular shapes, which can be modeled as mappings assigning letters from Σ or blank symbol $\#$ to discretized positions in the plane. Here, by *shape* we refer to those positions that are assigned letters (not the blank symbol). Isometric array languages are mostly described by some sort of isometric array grammars [98].

As we only encounter a special form of these grammars, namely, regular array grammars, we need not define any more details here. Finally, we will actually somehow abuse these devices to process arrays which have a rectangular shape, as they can be easily interpreted as isometric arrays.

Isometric regular array grammars (IRAG) [17, 114] have been introduced as the lowest level of the Chomsky hierarchy of grammars that describe two-dimensional languages. An isometric array consists of (finitely) many occurrences of symbols from Σ placed in the grid points (pixels) of \mathbb{Z}^2 (the discretized plane); the points of the plane which are not marked with elements of Σ are supposed to be marked with the blank symbol $\# \notin \Sigma$. Notice that the introduction of a blank symbol allows the description of pictures that are not of rectangular shape. These pictures are usually formalized (more generally) as mappings $\mathbb{Z}^2 \rightarrow \Sigma \cup \{\#\}$ with the understanding that symbols from Σ are assigned to at most finitely many positions (grid points). The collection of all such mappings (in other words, of all pictures), is denoted by Σ^{++} in this chapter, with the implicit understanding that $\# \notin \Sigma$ is reserved as a *background symbol*.

Sometimes, isometric arrays are considered identical if they can be transferred into each other by translation. The according equivalence classes are denoted by putting brackets around the arrays, array languages or families of such languages.

Now, we are going to formally relate the isometric arrays with non-isometric arrays. To this end, we identify some $m \times n$ matrix (a_{ij}) with entries from Σ , i. e., some element from Σ_+^+ , with the isometric array A with $A(i, j) = a_{ij}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$ and $A(i, j) = \#$ for all other cases. In this sense, we can think of Σ^{++} as being the set of all isometric rectangular arrays whose left upper non-blank entry is at coordinate $(1, 1)$.

Interestingly enough, also some sort of reverse embedding is possible. If $A \in \Sigma^{++}$ is an isometric array, then there is some smallest $m \times n$ rectangle such that

outside of this rectangle, only background symbols are attached to grid points via A . This (unique) rectangular-shaped array can be viewed as an element of $(\Sigma \cup \{\#\})_+^+$. In other words, we have an embedding $\text{emb}_1 : \Sigma_+^+ \rightarrow \Sigma^{++}$ and another embedding $\text{emb}_2 : \Sigma^{++} \rightarrow (\Sigma \cup \{\#\})_+^+$ such that:

- If $A \in \Sigma_+^+$, then $A = \text{emb}_2(\text{emb}_1(A))$.
- If $A \in \Sigma^{++}$ has a rectangular shape (concerning grid points labeled with symbols from Σ), if uppermost leftmost grid point of this rectangular shaped array has the coordinate $(1, 1)$, then $A = \text{emb}_1(\text{emb}_2(A))$.¹ Conversely, if $A \in \Sigma^{++}$ does not satisfy the above conditions, then $A \neq \text{emb}_1(\text{emb}_2(A))$.

For simplicity, we will identify Σ_+^+ with $\{A \in \Sigma^{++} \mid \text{emb}_1(A) \in \Sigma_+^+\}$.

We will now formally introduce the class of isometric grammars (and families of array languages) for this chapter.

Definition 54. An isometric regular array grammar G , or IRAG for short, is a quintuple $G = (N, \Sigma, P, S, \#)$, where N is the non-terminal alphabet, Σ is the terminal alphabet, P is the set of rules, $S \in N$ is the start symbol, and $\#$ is the blank symbol. Moreover, every rule from P is of the form

$$\#A \rightarrow Ba \quad , \quad A\# \rightarrow aB \quad , \quad \begin{array}{c} \# \\ A \end{array} \begin{array}{c} B \\ a \end{array} \quad , \quad \begin{array}{c} A \\ \# \end{array} \begin{array}{c} a \\ B \end{array} \quad , \quad \text{or} \quad A \rightarrow a \quad ,$$

where $A, B \in N$ and $a \in \Sigma$. The derivation of a grammar proceeds as follows:

- At the beginning, the whole discretized plane is filled with blank symbols.
- Then, the start symbol $S \in N$ is placed on some grid position, replacing $\#$. To be more precise, this means that an initial configuration is described by some mapping $\iota : \mathbb{Z}^2 \rightarrow \{S, \#\}$ such that $|\iota^-(S)| = 1$ (here ι^- denotes the inverse mapping of ι).
- Intermediately, we find that all non-blank symbols in the plane are terminal symbols but one, say, $A \in N$. Formally, this means that an intermediate configuration can be described by some mapping $\chi : \mathbb{Z}^2 \rightarrow N \cup \Sigma \cup \{\#\}$ satisfying $|\chi^-(N)| = 1$; in the case discussed in the following, we assume that $|\chi^-(A)| = 1$.

¹Note that we slightly abuse notation here, since, technically, some occurrences of $\#$ in $\text{emb}_1(\text{emb}_2(A))$ are background symbols, while others are actual terminals, already present in $\text{emb}_2(A)$; however, we interpret all these occurrences as background symbols.

- If the position left to A is blank, then we can apply a rule of the first listed type; this application replaces the blank symbol to the left of A by B and then A by a . This type of rule is therefore called a left movement.

Given χ , the successor configuration χ' hence satisfies:

$$\chi'(x, y) = \begin{cases} \chi(x, y) & \text{if } \chi(x, y) \neq A \\ a & \text{if } \chi(x, y) = A \\ B & \text{if } \chi(x, y) = \# \wedge \chi(x + 1, y) = A \end{cases}$$

The successor configurations χ' for the other types of rules informally described below can be formalized in a similar way.

- If the position right to A is blank, we can similarly apply the second type of rule. Applying such a rule implements a right movement.
 - If the position above A is blank, we can likewise apply the third type of rule. This means an upward movement.
 - If the position below A is blank, we can alternatively apply the fourth type of rule, which yields a downward movement.
- In any case, we can (if possible) apply the last type of rule; in that case A is replaced by a , so that none of the previously described rule applications are possible henceforth. That is, a terminal configuration can be seen as a mapping $\tau : \mathbb{Z}^2 \rightarrow \Sigma \cup \{\#\}$ satisfying $|\tau^-(\Sigma)| < \infty$.

Let $L(G)$ collect all isometric terminal arrays that can be derived by a finite sequence of rule applications in the described way. More formally, we write $\chi \Rightarrow \chi'$ if χ' is the successor configuration of χ . Then $L(G) = \{\tau : \mathbb{Z}^2 \rightarrow (\Sigma \cup \{\#\}) \mid \iota \Rightarrow^+ \tau\}$.

As the start position is arbitrary, any isometric terminal array $W : \mathbb{Z}^2 \rightarrow \Sigma \cup \{\#\} \in L(G)$ carries along an infinite number of other arrays $W' \in L(G)$ that can be obtained by translating W . Of course, this does not affect $[L(G)]$.

Definition 55. $\mathcal{L}_\Sigma(\text{IRAG}) = \{L(G) : G \text{ is an IRAG with terminal alphabet } \Sigma\}$.
 $L_{\text{Rect}}(G) = \{L(G) \cap \Sigma_+^+ : G \text{ is an IRAG with terminal alphabet } \Sigma\}$.
 $\mathcal{L}_{\text{Rect}, \Sigma}(\text{IRAG}) = \{L_{\text{Rect}}(G) : G \text{ is an IRAG with terminal alphabet } \Sigma\}$.

Let us now define four special types of IRAG that are defined by forbidding certain directions as we have introduced in [34]. Correspondingly, we obtain four types of families of picture languages.

Definition 56. Let G be some IRAG. If G contains no upwards (or downwards, or left, or right) movements, we face an \bar{U} -IRAG (or \bar{D} -IRAG, \bar{L} -IRAG, \bar{R} -IRAG, respectively). Let $X \in \{U, D, L, R\}$. Then,

$$\mathcal{L}_\Sigma(\bar{X}\text{-IRAG}) = \{L(G) : G \text{ is a } \bar{X}\text{-IRAG with terminal alphabet } \Sigma\}.$$

$$\mathcal{L}_{\text{Rect},\Sigma}(\bar{X}\text{-IRAG}) = \{L_{\text{Rect}}(G) : G \text{ is an } \bar{X}\text{-IRAG with terminal alphabet } \Sigma\}.$$

Example 21. Let us reconsider the set L_L of tokens L . A \bar{U} -IRAG $G = (N, \Sigma, P, S, \#)$ such that $L_{\text{Rect}}(G) = L_L$ is defined as follows:

- $N = \{S, A, B, E, F\}$,
- $\Sigma = \{x, \bullet\}$, and
- $P = \{S\# \rightarrow xA, A\# \rightarrow \bullet A, \#B \rightarrow B\bullet, E\# \rightarrow xE, \#F \rightarrow Fx,$
 $\begin{array}{c} A \quad \bullet \quad A \quad \bullet \quad B \quad x \quad B \quad x \\ \# \rightarrow B, \# \rightarrow F, \# \rightarrow S, \# \rightarrow E, E \rightarrow x, F \rightarrow x \end{array}\}.$

By way of contrast, the following grammar uses all four types of movements.

Example 22. Consider the array language L_\setminus the set of all square pictures of diagonal lines from the upper left corner to the lower right corner where the elements in the diagonal are 1 and the other elements are 0. An IRAG $G = (N, \Sigma, P, S, \#)$ such that $L_{\text{Rect}}(G) = L_\setminus$ is defined as follows: $N = \{S, A, B, C, H, E, F\}$, $\Sigma = \{0, 1\}$, $P = P_1 \cup P_2 \cup P_3$ where P_1 contains the following rules, sequentially numbered for the ease of presentation:

$$1: \begin{array}{c} \# \\ S \end{array} \rightarrow \begin{array}{c} A \\ 1 \end{array}, \quad 2: \begin{array}{c} \# \\ A \end{array} \rightarrow \begin{array}{c} A \\ 0 \end{array}, \quad 3: \#A \rightarrow B0, \quad 4: \#B \rightarrow B0, \quad 5: \begin{array}{c} B \\ \# \end{array} \rightarrow \begin{array}{c} 1 \\ F \end{array}, \quad 6: \begin{array}{c} C \\ \# \end{array} \rightarrow \begin{array}{c} 0 \\ C \end{array},$$

$$7: C\# \rightarrow 0H, \quad 8: H\# \rightarrow 0H, \quad 9: \begin{array}{c} \# \\ H \end{array} \rightarrow \begin{array}{c} E \\ 0 \end{array}, \quad a: \begin{array}{c} \# \\ E \end{array} \rightarrow \begin{array}{c} A \\ 1 \end{array}, \quad b: \begin{array}{c} F \\ \# \end{array} \rightarrow \begin{array}{c} 0 \\ C \end{array}, \quad c: S \rightarrow 1,$$

$$P_2 = \{d: F \rightarrow 0\}, \text{ and } P_3 = \{f: E \rightarrow 1\}.$$

Note: The rules from $P_1 \cup P_2$ are used to generate square arrays of even rows and columns and the rules from $P_1 \cup P_3$ are used to generate square arrays of odd rows and columns. The elements in L are the set of all pictures with equal number of rows and columns i. e., $m = n$. Hereafter we only say n and we have two cases for n . For the sake of presentation, we consider only $n = 5$ and $n = 6$ to illustrate the work of the grammar in these two cases, see Fig. 4.10. The arrows indicate the movements of the grammar when generating the sample, and the indices of these arrows refer to the rule labels.

However, also IRAGs with three directions only can be pretty tricky, in a sense, more than, say, BFAs. Consider the following example.

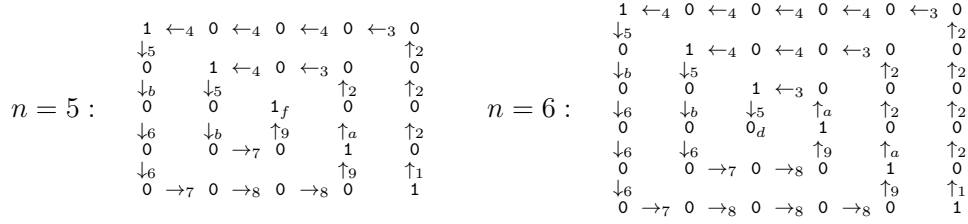


Figure 4.10: How an IRAG can generate L_{\setminus} .

Example 23. $G = (\{S, A, B, C\}, \{a, b, c\}, P, S, \#)$ with the following, relatively few, rules generates the following somehow complex language.

$$\begin{aligned}
L_{Rect}(G) &= ((\{b\}^+ \oplus \{a\}) \ominus (\{c\} \oplus \{c\}^+) \ominus (\{a\} \oplus \{a\}^+)) \\
&\cup (\{a\} \oplus \{a\}^+) \\
&\cup ((\{a\} \oplus \{a\}^+) \ominus (\{b\} \oplus \{b\}^+) \ominus (\{c\} \oplus \{c\}^+) \ominus (\{a\} \oplus \{a\}^+)) \\
&\cup (\{c\} \ominus \{a\}) \\
&\cup ((\{c\} \ominus \{a\}) \ominus (\{b\} \ominus \{c\} \ominus \{a\})_+)
\end{aligned}$$

Here, P contains:

- *Start rules:* $S\# \rightarrow aA$, $\#S \rightarrow Ba$, $\frac{S}{\#} \rightarrow \frac{c}{A}$;
- *Terminating rule:* $A \rightarrow a$;
- *Row-moving rules:* $A\# \rightarrow aA$, $\#A \rightarrow Aa$, $\#B \rightarrow Bb$, $C\# \rightarrow cC$;
- *Column-moving rules:* $\frac{A}{\#} \rightarrow \frac{a}{B}$, $\frac{B}{\#} \rightarrow \frac{b}{C}$, $\frac{C}{\#} \rightarrow \frac{c}{A}$.

We now introduce three mappings on the set P of an IRAG $G = (N, \Sigma, P, S, \#)$. To this end, it is convenient to think of P being partitioned into P_r (collecting all right movements), P_d (downward movements), P_ℓ (left movements), P_u (upward movements) and P_t (terminating rules).

- R_v acts as the identity on $P_d \cup P_u \cup P_t$. For $p = \#A \rightarrow Ba \in P_\ell$, we give $R_v(p) = A\# \rightarrow aB$. Hence, $R_v(P_\ell) = (R_v(P))_r$. Likewise, for $p' = A\# \rightarrow aB \in P_r$, we set $R_v(p') = \#A \rightarrow Ba$. Hence, $R_v(P_r) = (R_v(P))_\ell$.
- R_h acts as the identity on $P_r \cup P_\ell \cup P_t$. For $p = \frac{\#}{A} \rightarrow \frac{B}{a} \in P_u$, we set $R_h(p) = \frac{A}{\#} \rightarrow \frac{a}{B}$. Hence, $R_h(P_u) = (R_h(P))_d$. Similarly, downward movements are mapped onto upward movements.

- T is the identity on P_t . For $p = \#^A \rightarrow B^a \in P_\ell$, we introduce $T(p) = \#^{\rightarrow A} \rightarrow B^a$. This way, $T(P_\ell) = (T(P))_u$. Similarly, upward movements of P are mapped onto left movements of $T(P)$, right movements of P are mapped onto downwards movements of $T(P)$ and vice versa.

We further transfer these operations R_v , R_h and T to grammars and write, for instance, $T(G)$ for the grammar $(N, \Sigma, T(P), S, \#)$ obtained from the IRAG $G = (N, \Sigma, P, S, \#)$. By following the derivation of a grammar step-by-step, it is straightforward to conclude the following lemma.

Lemma 32. *Let $G = (N, \Sigma, P, S, \#)$ be an IRAG. Then,*

- $R_v(L_{Rect}(G)) = L_{Rect}(R_v(G))$,
- $R_h(L_{Rect}(G)) = L_{Rect}(R_h(G))$, and
- $T(L_{Rect}(G)) = L_{Rect}(T(G))$.

These simple observations allow us to conclude some nice properties of families of languages of rectangular-shaped arrays that can be described by IRAGs.

Corollary 10. $\mathcal{L}_{Rect, \Sigma}(\text{IRAG})$ is closed under the reflection operations R_v , R_h and T .

Corollary 11. *The reflection operations translate language classes as follows.*

- $R_v(\mathcal{L}_{Rect, \Sigma}(\overline{R}\text{-IRAG})) = \mathcal{L}_{Rect, \Sigma}(\overline{L}\text{-IRAG})$,
 $R_v(\mathcal{L}_{Rect, \Sigma}(\overline{L}\text{-IRAG})) = \mathcal{L}_{Rect, \Sigma}(\overline{R}\text{-IRAG})$,
 $\mathcal{L}_{Rect, \Sigma}(\overline{U}\text{-IRAG})$ is closed under R_v ,
 $\mathcal{L}_{Rect, \Sigma}(\overline{D}\text{-IRAG})$ is closed under R_v ,
- $R_h(\mathcal{L}_{Rect, \Sigma}(\overline{U}\text{-IRAG})) = \mathcal{L}_{Rect, \Sigma}(\overline{D}\text{-IRAG})$,
 $R_h(\mathcal{L}_{Rect, \Sigma}(\overline{D}\text{-IRAG})) = \mathcal{L}_{Rect, \Sigma}(\overline{U}\text{-IRAG})$,
 $\mathcal{L}_{Rect, \Sigma}(\overline{L}\text{-IRAG})$ is closed under R_h ,
 $\mathcal{L}_{Rect, \Sigma}(\overline{R}\text{-IRAG})$ is closed under R_h ,
- $T(\mathcal{L}_{Rect, \Sigma}(\overline{L}\text{-IRAG})) = \mathcal{L}_{Rect, \Sigma}(\overline{U}\text{-IRAG})$,
 $T(\mathcal{L}_{Rect, \Sigma}(\overline{U}\text{-IRAG})) = \mathcal{L}_{Rect, \Sigma}(\overline{L}\text{-IRAG})$,
 $T(\mathcal{L}_{Rect, \Sigma}(\overline{D}\text{-IRAG})) = \mathcal{L}_{Rect, \Sigma}(\overline{R}\text{-IRAG})$,
 $T(\mathcal{L}_{Rect, \Sigma}(\overline{R}\text{-IRAG})) = \mathcal{L}_{Rect, \Sigma}(\overline{D}\text{-IRAG})$.

Theorem 25. $\mathcal{L}_{\Sigma}(\text{BFA}) = \mathcal{L}_{Rect, \Sigma}(\overline{U}\text{-IRAG})$.

Proof. We first show that $\mathcal{L}_\Sigma(\text{BFA}) \subseteq \mathcal{L}_{\text{Rect},\Sigma}(\overline{U}\text{-IRAG})$. Let $M = (Q, \Sigma, R, (s, r), F, \#, \square)$ be some BFA. W.l.o.g., according to Lemma 30, we can assume that M is a BFA in DANF; so, the second component of the state alphabet gives the direction of the movements, and we start with a right movement. We remove useless states from M so that M has only useful states.

Now define an $\overline{U}\text{-IRAG}$, $G = (N, \Sigma, P, S, \#)$, with $L(M) = L_{\text{Rect}}(G)$, in three steps. As first step we extract LEFT, RIGHT movements and terminal rules. As second step we extract the DOWNWARD movement rules of the target $\overline{U}\text{-IRAG}$ from the given BFA M by combining the non- $\#$ rule with the $\#$ rule of the BFA M . Finally, we collect all rules in steps 1 and 2 and eliminate the useless non-terminals and rules, and we define the $\overline{U}\text{-IRAG}$ G such that $L(M) = L_{\text{Rect}}(G)$. Let us write the three steps formally as follows:

First Step: For the right and left movements and for the terminating rules we include the following production rules into P :

- $\#(p, \ell) \rightarrow (q, \ell)a \in P$ for all $(p, \ell)a \rightarrow (q, \ell) \in R$, $a \in \Sigma$,
- $(p, r)\# \rightarrow a(q, r) \in P$ for all $(p, r)a \rightarrow (q, r) \in R$, $a \in \Sigma$,
- $(p, d) \rightarrow a \in P$ for all $(p, d)a \rightarrow (f, d) \in R$, $a \in \Sigma$, $(f, d) \in F$.

Second Step: For direction-changing steps, we add DOWNWARD movement rules. As can be seen, they combine two rules of the BFA, as the grammar never ‘reads’ the border marker $\#$.

- $\begin{matrix} (p,r) \\ \# \end{matrix} \rightarrow \begin{matrix} a \\ (q',\ell) \end{matrix} \in P$ for all $\{(p, r)a \rightarrow (q, r), (q, r)\# \rightarrow (q', \ell)\} \subseteq R$;
- $\begin{matrix} (p,\ell) \\ \# \end{matrix} \rightarrow \begin{matrix} a \\ (q',r) \end{matrix} \in P$ for all $\{(p, \ell)a \rightarrow (q, \ell), (q, \ell)\# \rightarrow (q', r)\} \subseteq R$.

Third Step: Define required $\overline{U}\text{-IRAG}$ $G = (N, \Sigma, P, S, \#)$, N is constructed from Q by possibly eliminating useless non-terminals, $S = (s, r)$, and P collects all rules described in the first and second step; again P might skip useless rules.

A computation of the BFA is imitated by a derivation of the target grammar $\overline{U}\text{-IRAG}$ step by step. At any stage of the BFA, the current state of the BFA is matched with the current non-terminal of the simulating $\overline{U}\text{-IRAG}$ as in Fig. 4.11. The merge of configurations of BFA and $\overline{U}\text{-IRAG}$ in this figure yields the input array, where \square in BFA and $\#$ in $\overline{U}\text{-IRAG}$ are transparent during merging. Hence $\mathcal{L}_\Sigma(\text{BFA}) \subseteq \mathcal{L}_{\text{Rect},\Sigma}(\overline{U}\text{-IRAG})$.

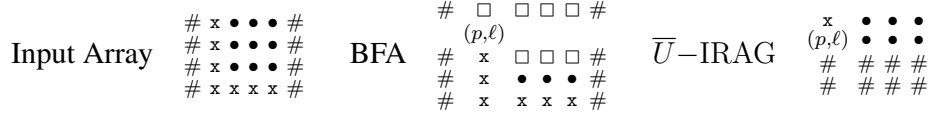


Figure 4.11: How array processing of automata and grammars complement.

$\mathcal{L}_{Rect, \Sigma}(\overline{U}\text{-IRAG})$ contains languages that can be generated by IRAGs which have rules with the directions R, L, D . Considering the non-terminals of such grammar as states, a BFA can simulate the movements, since the arrays in this language family are of rectangular shape. There is one technicality that needs attention, though: The derivation of a \overline{U} -IRAG could start either in the right or in the left direction, and there is even the possibility to create single columns by moving down straight away.

For a more formal argument, consider the \overline{U} -IRAG $G = (N, \Sigma, P, S, \#)$. Let us first memorize the first direction that was taken by constructing $G_d = (N \times \{L, R, D\} \cup \{S\}, \Sigma, P_d, S, \#)$ by defining:

- $S\# \rightarrow \mathbf{x}(A, R) \in P_d$ if $S\# \rightarrow \mathbf{x}A \in P$,
- $\#S \rightarrow (A, L)\mathbf{x} \in P_d$ if $\#S \rightarrow A\mathbf{x} \in P$,
- $\frac{S}{\#} \rightarrow (\frac{\mathbf{x}}{A}, D) \in P_d$ if $\frac{S}{\#} \rightarrow \frac{\mathbf{x}}{A} \in P$.

From now on, this start direction is preserved by setting $(A, X)\# \rightarrow \mathbf{x}(B, X) \in P_d$ for $X \in \{L, R, D\}$ if $A\# \rightarrow \mathbf{x}B \in P$, and similarly for the other (left, down) directions and for the terminating rules. Of course, this might introduce some rules that are not reachable from S that we can easily eliminate now.

We can classify rules of G_d collected in P_d into those having the start direction (consistently) attached to the non-terminals that occur in it, leading to the rule sets $P_{d,X}$ for $X \in \{L, R, D\}$. Based on this we can form grammars $G_{d,X} = (N \times \{X\} \cup \{S\}, \Sigma, P_{d,X}, S, \#)$. Clearly by construction, $L(G) = L(G_d) = L(G_{d,L}) \cup L(G_{d,R}) \cup L(G_{d,D})$. Also $L_{Rect}(G) = L_{Rect}(G_d) = L_{Rect}(G_{d,L}) \cup L_{Rect}(G_{d,R}) \cup L_{Rect}(G_{d,D})$.

In the following arguments, we can assume that the IRAG that we consider can start only in one direction: left, right, or down. We can easily modify the non-terminal alphabets, so they only contain the start symbol as a common symbol, and as the non-terminal symbols will correspond to the states of the BFAs that we construct (plus one final state f), it is easy to see that we can form one BFA

out of the three BFAs that are obtained from these three IRAGs, starting out into different directions.

Case 1: Downward Start. Consider $G_{d,D}$ as in the previous construction. This is the simplest case, as we know that, in order to describe a rectangular array, $G_{d,D}$ can only perform downwards movements or terminating rules. We can simulate a rule $\frac{S}{\#} \rightarrow (A,D) \in P_{d,D}$ by the two rules $Sx \rightarrow A'$ and $A'\# \rightarrow A$. Similarly, $\frac{(A,D)}{\#} \rightarrow (B,D) \in P_{d,D}$ is imitated by the two rules $Ax \rightarrow B'$ and $B'\# \rightarrow B$. A terminating rule $(A, D) \rightarrow a$ corresponds to a rule $Aa \rightarrow f$, where f is the only final state of the BFA. Let us call the BFA simulating the downward start M_D .

Let us have the following preparatory step, before we start the remaining two cases for the Right and Left Starts:

Interludium: Right and Left Starts. As a preparatory step, let us first modify $G_{d,R}$ and $G_{d,L}$ as follows, leading to $\overline{G}_{d,R}$ and $\overline{G}_{d,L}$. The (new) non-terminals (apart from the start symbol) look like $(A, 0, R)$ and $(A, 1, R)$ for non-terminals (A, R) of $G_{d,R}$ and similarly with $G_{d,L}$. Start rules have $(A, 0, X)$ on their right-hand side instead of (A, X) in $G_{d,X}$, $X \in \{L, R\}$. Similarly, terminating rules have $(A, 0, X)$ or $(A, 1, X)$ on their left-hand side instead of (A, X) in $G_{d,X}$.

The downward-moving rules are adapted as follows. $\frac{(A,X)}{\#} \rightarrow (B,X) \in P_{d,X}$ is replaced by $\frac{(A,0,X)}{\#} \rightarrow (B,1,X)$ and $\frac{(A,1,X)}{\#} \rightarrow (B,0,X)$. The switch between 0 and 1 in the second component is essential, as here we store if the grammar is working on an odd- or even-numbered row. Moreover, $\overline{G}_{d,R}$ contains the following left- and right-moving rules. Notice that now the information if the grammar works on an odd- or even-numbered row is simply maintained.

- $(A, 0, R)\# \rightarrow a(B, 0, R) \in P'_{d,R}$ if $(A, R)\# \rightarrow a(B, R) \in P_d$,
- $\#(A, 1, R) \rightarrow (B, 1, R)a \in P'_{d,R}$ if $\#(A, R) \rightarrow (B, R)a \in P_d$.

Similarly, $\overline{G}_{d,L}$ contains the following left- and right-moving rules.

- $(A, 1, L)\# \rightarrow a(B, 1, L) \in P'_{d,L}$ if $(A, L)\# \rightarrow a(B, L) \in P_d$,
- $\#(A, 0, L) \rightarrow (B, 0, L)a \in P'_{d,L}$ if $\#(A, L) \rightarrow (B, L)a \in P_d$.

Again, we can simplify the obtained grammars by deleting unreachable rules.

Case 2: Right start. Now, the processing corresponds exactly to that of BFAs, as we are interested only in rectangular-shaped arrays. This means, in particular, that a sequence of right moves must be followed by one downward move, and then a sequence of left moves follows, etc. The only possibly critical thing is observed at

the borders: Instead of a (direct) downward movement, the simulating BFA again has to first sense the current symbol and then the background symbol; this can be achieved as in Case 1. We therefore omit details here. However, notice that the resulting BFA is already direction-aware. The states of the form $(A, 0, R)$ move right, and the states of the form $(A, 1, R)$ move left. Hence, we can obtain M_R from $G'_{d,R}$.

Case 3: Left start. Here, a sequence of left moves should be followed by one downward move, and then a sequence of right moves and so on. This can be simulated with the idea of the mirror-image construction known for regular string languages, but this idea has to be combined with the construction from previous case.

More formally, consider $G'_{d,L} = (N'_{d,L}, \Sigma, P'_{d,L}, S, \#)$ obtained from Interludium Step. We first construct a grammar $G_{d,L}$ that generates $L_{d,L} = L_{Rect}(G'_{d,L})$ by only starting with right-moves. $G_{d,L} = (N_{d,L}, \Sigma, P_{d,L}, S'_L, \#)$ is defined by $N_{d,L} = ((N \cup \{\Phi\}) \times N'_{d,L} \times N) \cup \{S'_L\}$, where $S'_L \notin N$ is a new start symbol and $\Phi \notin N$ is a new symbol indicating (guessing) the end of a derivation in the current row.

$$\begin{aligned}
P_{d,L} = & \left\{ S'_L \# \rightarrow \mathbf{x}(Y, X, 0, L, S) \mid \binom{(X,0,L)}{\#} \rightarrow (Y, \mathbf{x}, L) \in P'_{d,L} \right\} \\
& \cup \left\{ S'_L \# \rightarrow \mathbf{x}(\Phi, X, 0, L, S) \mid (X, 0, L) \rightarrow \mathbf{x} \in P'_{d,L} \right\} \\
& \cup \left\{ (Y, X, 0, L, S) \# \rightarrow \mathbf{x}(Y, Z, 0, L, S) \mid \right. \\
& \quad \left. \#(Z, 0, L) \rightarrow (X, 0, L) \mathbf{x} \in P'_{d,L}, Y = \Phi \vee (Y, 1, L) \in N'_{d,L} \right\} \\
& \cup \left\{ (Y, X, 0, L, A) \# \rightarrow \mathbf{x}(Y, Z, 0, L, A) \mid \right. \\
& \quad \left. \#(Z, 0, L) \rightarrow (X, 0, L) \mathbf{x} \in P'_{d,L}, (A, 0, L) \in N'_{d,L}, Y = \Phi \vee (Y, 1, L) \in N'_{d,L} \right\} \\
& \cup \left\{ \#(Y, X, 1, L, A) \rightarrow (Y, Z, 1, L, A) \mathbf{x} \mid \right. \\
& \quad \left. (Z, 1, L) \# \rightarrow \mathbf{x}(X, 1, L) \in P'_{d,L}, (A, 1, L) \in N'_{d,L}, Y = \Phi \vee (Y, 0, L) \in N'_{d,L} \right\} \\
& \cup \left\{ \binom{(Y, X, 0, L, S)}{\#} \rightarrow (B, A, \mathbf{x}, L, Y) \mid \#S \rightarrow (X, 0, L) \mathbf{x} \in P'_{d,L}, (A, 1, L) \in N'_{d,L}, \right. \\
& \quad \left. (Y, 1, L) \in N'_{d,L}, (B = \Phi \vee (B, 1, L) \in N'_{d,L}) \right\} \\
& \cup \left\{ \binom{(Y, X, 0, L, Z)}{\#} \rightarrow (B, A, \mathbf{x}, L, Y) \mid \#(Z, 0, L) \rightarrow (X, 0, L) \mathbf{x} \in P'_{d,L}, (A, 1, L) \in N'_{d,L}, \right. \\
& \quad \left. (Y, 1, L) \in N'_{d,L}, (B = \Phi \vee (B, 1, L) \in N'_{d,L}) \right\} \\
& \cup \left\{ \binom{(Y, X, 1, L, Z)}{\#} \rightarrow (B, A, 0, L, Y) \mid (Z, 1, L) \# \rightarrow \mathbf{x}(X, 1, L) \in P'_{d,L}, (A, 0, L) \in N'_{d,L}, \right. \\
& \quad \left. (Y, 0, L) \in N'_{d,L}, (B = \Phi \vee (B, 0, L) \in N'_{d,L}) \right\} \\
& \cup \left\{ (\Phi, X, 0, L, S) \rightarrow \mathbf{x} \mid \#S \rightarrow (X, 0, L) \mathbf{x} \in P'_{d,L} \right\} \\
& \cup \left\{ (\Phi, X, 0, L, Z) \rightarrow \mathbf{x} \mid \#(Z, 0, L) \rightarrow (X, 0, L) \mathbf{x} \in P'_{d,L} \right\} \\
& \cup \left\{ (\Phi, X, 1, L, Z) \rightarrow \mathbf{x} \mid (Z, 1, L) \# \rightarrow \mathbf{x}(X, 1, L) \in P'_{d,L} \right\}.
\end{aligned}$$

Notice that $G_{d,L}$ always starts into the right direction. More precisely, the rule $S'_L \# \rightarrow \mathbf{x}(Y, X, 0, L, S)$ maintains the following pieces of information: (1) As

it produces the symbol x in the leftmost upper corner of the image, it did in fact simulate one part of the rule $\frac{(X,0,L)}{\#} \rightarrow (Y,\overset{x}{1},L) \in P'_{d,L}$. (2) As the second part of simulation would be to move down, continue with non-terminal Y , coming from non-terminal X ; these information are stored in first two components of the new non-terminal $(Y, X, 0, L, S)$. (3) As in the mentioned mirror-image construction, we have to also store as the target non-terminal the start symbol of the simulated grammar, which is the purpose of the last component of the new non-terminal.

The mirror-image construction is fully reflected in rules that are of the form $(Y, X, 0, L, S)\# \rightarrow x(Y, Z, 0, L, S)$, where $\#(Z, 0, L) \rightarrow (X, 0, L)x \in P'_{d,L}$; originally, the grammar was generating from left to right, switching from Z to X and producing x , but in the simulating grammar, the generation is from right to left, switching from X to Z in the second component of the non-terminal. The simulation of the generation of the first row is completed with $\frac{(Y,X,0,L,S)}{\#} \rightarrow (B,A,\overset{x}{1},L,Y)$ for $\#S \rightarrow (X, 0, L)x \in P'_{d,L}$.

Apart from simulating the left move of the simulated grammar by a right move as described with the previous case of rules, the simulating grammar also switches into processing the next row, guessing new symbols A and B to be verified later, and setting the previously remembered symbol Y as the new target symbol to be reached at the end of reading the second row.

Recall that this symbol Y was part of the very first rule simulation of simulating grammar, and this ensures that the correct simulation of this first guessed rule (of $P'_{d,L}$) will be checked after reaching the left border again (in the second row). The third component of the new non-terminal serves to indicate the parity of the row that is worked in, as was the case for the simulated grammar.

The simulation of left moves of the original grammar is similarly seen by right moves of the simulating grammar. In the end, it is guessed that the simulation terminates, by guessing Φ . This allows to end the simulation. For instance, if it was (already) guessed that only a single-row picture is generated, the derivation of the simulating grammar stops with applying a rule $(\Phi, X, 0, L, S) \rightarrow x$ for $\#S \rightarrow (X, 0, L)x \in P'_{d,L}$.

In the end, we can apply the construction of Case 2 in order to obtain the BFA M_L from $G_{d,L}$.

Final Construction. As the state alphabets of M_D , M_R , and M_L can be thought of having only the start state (after renaming) and the final state f in common, we can

build the BFA simulating the originally given IRAG by simply merging the rule sets of the three automata M_D , M_R , and M_L . Hence, we also have $\mathcal{L}_\Sigma(\text{BFA}) \supseteq \mathcal{L}_{\text{Rect},\Sigma}(\overline{U}\text{-IRAG})$. \square

As a consequence of Theorem 25 and Cor. 11 we obtain:

Corollary 12. *For each alphabet Σ ,*

$$T(\mathcal{L}_\Sigma(\text{BFA})) = \mathcal{L}_{\text{Rect},\Sigma}(\overline{L}\text{-IRAG}).$$

Illustration 7. *Let us now illustrate Theorem 25 by reconsidering the d-BFA M in Fig. 4.4. As per the construction in Theorem 25, let us now formally define an \overline{U} -IRAG $G = (N, \Sigma, P, S, \#)$ with $L(M) = L_{\text{Rect}}(G)$ in three steps. We focus on presenting the rules, giving reasons based on the transitions of M . As $(s_3, \underline{\ell})$ is a useless non-terminal, we omit it and all rules that would produce it.*

1. $\#(s_2, \underline{\ell}) \rightarrow (s_2, \underline{\ell})\bullet$ and $\#(s_4, \underline{\ell}) \rightarrow (s_4, \underline{\ell})\mathbf{x}$ are the left movements, due to the transitions $(s_2, \underline{\ell})\bullet \rightarrow (s_2, \underline{\ell})$ and $(s_4, \underline{\ell})\mathbf{x} \rightarrow (s_4, \underline{\ell})$.
2. $(s, \underline{r})\# \rightarrow \mathbf{x}(s_5, \underline{r})$, $(s_5, \underline{r})\# \rightarrow \bullet(s_1, \underline{r})$, $(s_1, \underline{r})\# \rightarrow \bullet(s_1, \underline{r})$ and $(s_4, \underline{r})\# \rightarrow \mathbf{x}(s_4, \underline{r})$ are all right movements, based on $(s, \underline{r})\mathbf{x} \rightarrow (s_5, \underline{r})$, etc.
3. $(s_4, \underline{r}) \rightarrow \mathbf{x}$ and $(s_4, \underline{\ell}) \rightarrow \mathbf{x}$ are the two terminating rules, due to the two final states of the given d-BFA.
4. $\begin{array}{c} (s_5, \underline{r}) \rightarrow \bullet \\ \# \rightarrow (s_2, \underline{\ell}) \\ (s_2, \underline{\ell}) \rightarrow \mathbf{x} \\ \# \rightarrow (s_4, \underline{r}) \end{array}, \begin{array}{c} (s_5, \underline{r}) \rightarrow \bullet \\ \# \rightarrow (s_4, \underline{\ell}) \\ (s_4, \underline{\ell}) \rightarrow \mathbf{x} \\ \# \rightarrow (s_2, \underline{\ell}) \end{array}, \begin{array}{c} (s_1, \underline{r}) \rightarrow \bullet \\ \# \rightarrow (s_4, \underline{\ell}) \\ (s_4, \underline{\ell}) \rightarrow \mathbf{x} \\ \# \rightarrow (s_2, \underline{\ell}) \end{array}, \begin{array}{c} (s_1, \underline{r}) \rightarrow \bullet \\ \# \rightarrow (s_2, \underline{\ell}) \\ (s_2, \underline{\ell}) \rightarrow \mathbf{x} \\ \# \rightarrow (s, \underline{r}) \end{array}$ and $(s_2, \underline{\ell}) \rightarrow \mathbf{x}$ are the downwards movements, due to combining pairs of transitions like $(s_5, \underline{r})\bullet \rightarrow (s, \underline{r})$, $(s, \underline{r})\# \rightarrow (s_2, \underline{\ell})$, etc.

Illustration 8. *Let us reconsider G in Example 23 to illustrate reverse direction of Theorem 25. Our aim is to find a BFA M with $L(M) = L_{\text{Rect}}(G)$, following the construction of the reverse direction of Theorem 25. Let us first construct G_d as follows: $G_d = (N_d, \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}, P_d, S, \#)$ where $N_d = \{A, B, C\} \times \{L, R, D\} \cup$*

$\{S\}$, i. e., N_d has 10 non-terminals and P_d has the following 27 rules.

$$\begin{aligned}
P_d = & \left\{ \#S \rightarrow (B, L)\mathbf{a}, S\# \rightarrow \mathbf{a}(A, R), \overset{S}{\#} \rightarrow \overset{c}{(A, D)} \right\} \\
& \cup \{(A, L)\# \rightarrow \mathbf{a}(A, L), (A, R)\# \rightarrow \mathbf{a}(A, R), (A, D)\# \rightarrow \mathbf{a}(A, D)\} \\
& \cup \{\#(A, L) \rightarrow (A, L)\mathbf{a}, \#(A, R) \rightarrow (A, R)\mathbf{a}, \#(A, D) \rightarrow (A, D)\mathbf{a}\} \\
& \cup \{\#(B, L) \rightarrow (B, L)\mathbf{b}, \#(B, R) \rightarrow (B, R)\mathbf{b}, \#(B, D) \rightarrow (B, D)\mathbf{b}\} \\
& \cup \{\#(C, L) \rightarrow (C, L)\mathbf{c}, \#(C, R) \rightarrow (C, R)\mathbf{c}, \#(C, D) \rightarrow (C, D)\mathbf{c}\} \\
& \cup \left\{ \begin{array}{ccc} (A, L) & \mathbf{a} & (A, R) \\ \# & \rightarrow & (B, L) \end{array}, \begin{array}{ccc} (A, R) & \mathbf{a} & (A, D) \\ \# & \rightarrow & (B, R) \end{array}, \begin{array}{ccc} (A, D) & \mathbf{a} & \\ \# & \rightarrow & (B, D) \end{array} \right\} \\
& \cup \left\{ \begin{array}{ccc} (B, L) & \mathbf{b} & (B, R) \\ \# & \rightarrow & (C, L) \end{array}, \begin{array}{ccc} (B, R) & \mathbf{b} & (B, D) \\ \# & \rightarrow & (C, R) \end{array}, \begin{array}{ccc} (B, D) & \mathbf{b} & \\ \# & \rightarrow & (C, D) \end{array} \right\} \\
& \cup \left\{ \begin{array}{ccc} (C, L) & \mathbf{c} & (C, R) \\ \# & \rightarrow & (A, L) \end{array}, \begin{array}{ccc} (C, R) & \mathbf{c} & (C, D) \\ \# & \rightarrow & (A, R) \end{array}, \begin{array}{ccc} (C, D) & \mathbf{c} & \\ \# & \rightarrow & (A, D) \end{array} \right\} \\
& \cup \{(A, L) \rightarrow \mathbf{a}, (A, R) \rightarrow \mathbf{a}, (A, D) \rightarrow \mathbf{a}\}.
\end{aligned}$$

We also have $P_{d,L}$, $P_{d,R}$ and $P_{d,D}$ colored in blue, red and violet, respectively, for the grammars $G_{d,L}$, $G_{d,R}$ and $G_{d,D}$, respectively, that classify the initial rule used by G as being a *leftward*, *rightward*, or *downward* movement, respectively. Clearly by construction $L(G) = L(G_d) = L(G_{d,L}) \cup L(G_{d,R}) \cup L(G_{d,D})$ and also $L_{\text{Rect}}(G) = L_{\text{Rect}}(G_d) = L_{\text{Rect}}(G_{d,L}) \cup L_{\text{Rect}}(G_{d,R}) \cup L_{\text{Rect}}(G_{d,D})$. Finally, we form one BFA M_{LRD} out of the three BFAs M_L , M_R and M_D that are obtained from the three \bar{U} -IRAGs: $G_{d,L}$, $G_{d,R}$ and $G_{d,D}$, starting in different directions.

Downward Start: Consider $G_{d,D}$. Recall $G_{d,D}$ can only execute downward movements or terminating rules to describe a rectangular array. We simulate the rules from $P_{d,D}$ as follows with transitions from M_D , collected in R_D :

- $Sc \rightarrow A' \in R_D$ and $A'\# \rightarrow A \in R_D$ since $\overset{S}{\#} \rightarrow \overset{c}{(A, D)} \in P_{d,D}$,
- $Aa \rightarrow B' \in R_D$ and $B'\# \rightarrow B \in R_D$ since $\begin{array}{ccc} (A, D) & \mathbf{a} & \\ \# & \rightarrow & (B, D) \end{array} \in P_{d,D}$,
- $Bb \rightarrow C' \in R_D$ and $C'\# \rightarrow C \in R_D$ since $\begin{array}{ccc} (B, D) & \mathbf{b} & \\ \# & \rightarrow & (C, D) \end{array} \in P_{d,D}$,
- $Cc \rightarrow D' \in R_D$ and $D'\# \rightarrow A \in R_D$ since $\begin{array}{ccc} (C, D) & \mathbf{c} & \\ \# & \rightarrow & (A, D) \end{array} \in P_{d,D}$,
- $Aa \rightarrow f \in R_D$ since $(A, D) \rightarrow \mathbf{a} \in P_{d,D}$.

At this stage we have the BFA $M_D = (Q_D, \Sigma, R_D, S, \{f\}, \#, \square)$ as in Fig. 4.12. Notice that

$$L(M_D) = (\{c\} \ominus \{a\}) \cup ((\{c\} \ominus \{a\}) \ominus (\{b\} \ominus \{c\} \ominus \{a\})_+).$$

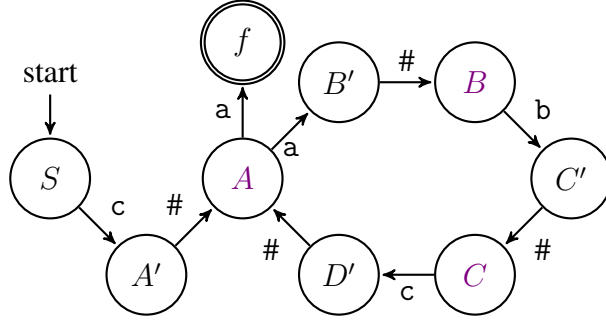


Figure 4.12: BFA M_D that accepts the language $L_{Rect}(G_{d,D})$ in Illustration 8

Interludium: Let us modify $G_{d,L}$, $G_{d,R}$ to obtain $G'_{d,L}$ and $G'_{d,R}$ as follows:

The non-terminals (apart from start symbol) look like $(A, 0, X)$ and $(A, 1, X)$ for $(A, X) \in N_{d,R} \cup N_{d,L}$. More explicitly, $G'_{d,L} = (N'_{d,L}, \Sigma, P'_{d,L}, S, \#)$ where $N'_{d,L} = \{A, B, C\} \times \{0, 1\} \times \{L\} \cup S$, i. e., $N'_{d,L}$ has 7 non-terminals and $P'_{d,L}$ has 13 rules, but when we follow the construction given in Theorem 25, we delete unreachable rules and we obtain $Reach - P'_{d,L}$, that has only the reachable rules. Both $P'_{d,L}$ and $Reach - P'_{d,L}$ are explicitly written as follows:

$$\begin{aligned}
P'_{d,L} &= \{ \#S \rightarrow (B, 0, L)a \} \\
&\cup \{ (A, 0, L) \rightarrow a, (A, 1, L) \rightarrow a \} \\
&\cup \left\{ \begin{array}{l} (A,0,L) \rightarrow a \quad (A,1,L) \rightarrow a \\ \# \rightarrow (B,1,L)', \quad \# \rightarrow (B,0,L) \end{array} \right\} \\
&\cup \left\{ \begin{array}{l} (B,0,L) \rightarrow b \quad (B,1,L) \rightarrow b \\ \# \rightarrow (C,1,L)', \quad \# \rightarrow (C,0,L) \end{array} \right\} \\
&\cup \left\{ \begin{array}{l} (C,0,L) \rightarrow c \quad (C,1,L) \rightarrow c \\ \# \rightarrow (A,1,L)', \quad \# \rightarrow (A,0,L) \end{array} \right\} \\
&\cup \{ (A, 1, L)\# \rightarrow a(A, 1, L), (C, 1, L)\# \rightarrow c(C, 1, L) \} \\
&\cup \{ \#(A, 0, L) \rightarrow (A, 0, L)a, \#(B, 0, L) \rightarrow (B, 0, L)b \}.
\end{aligned}$$

$$\begin{aligned}
Reach - P'_{d,L} &= \{ \#S \rightarrow (B, 0, L)a \} \cup \{ (A, 0, L) \rightarrow a \} \\
&\cup \left\{ \begin{array}{l} (B,0,L) \rightarrow b \\ \# \rightarrow (C,1,L) \end{array} \right\} \cup \left\{ \begin{array}{l} (C,1,L) \rightarrow c \\ \# \rightarrow (A,0,L) \end{array} \right\} \\
&\cup \{ (C, 1, L)\# \rightarrow c(C, 1, L) \} \\
&\cup \{ \#(A, 0, L) \rightarrow (A, 0, L)a, \#(B, 0, L) \rightarrow (B, 0, L)b \}.
\end{aligned}$$

Moreover, $G'_{d,R} = (N'_{d,R}, \Sigma, P'_{d,R}, S, \#)$, where $N'_{d,R} = \{A, B, C\} \times \{0, 1\} \times \{R\} \cup S$, i. e., $N'_{d,R}$ also has 7 non-terminals and $P'_{d,R}$ has 13 rules but when we

follow the construction given in Theorem 25, we delete unreachable rules. Only $Reach - P'_{d,R}$ is made explicit as follows:

$$\begin{aligned}
Reach - P'_{d,R} &= \{S\# \rightarrow a(A, 0, R)\} \cup \{(A, 0, R) \rightarrow a, (A, 1, R) \rightarrow a\} \\
&\cup \left\{ \begin{array}{c} (A,0,R) \\ \# \end{array} \rightarrow \begin{array}{c} a \\ (B,1,R) \end{array} \right\} \cup \left\{ \begin{array}{c} (B,1,R) \\ \# \end{array} \rightarrow \begin{array}{c} b \\ (C,0,R) \end{array} \right\} \\
&\cup \left\{ \begin{array}{c} (C,0,R) \\ \# \end{array} \rightarrow \begin{array}{c} c \\ (A,1,R) \end{array} \right\} \\
&\cup \{(A, 0, R)\# \rightarrow a(A, 0, R), (C, 0, R)\# \rightarrow c(C, 0, R)\} \\
&\cup \{\#(A, 1, R) \rightarrow (A, 1, R)a, \#(B, 1, R) \rightarrow (B, 1, R)b\}.
\end{aligned}$$

Right Start: From grammar $G'_{d,R}$ that we have constructed above, let us consider only the reachable rules $Reach-P'_{d,R}$ and let us simulate those reachable rules to obtain the BFA M_R as in Fig. 4.13. Here we can note that states that have 0 in their triple move right and that the states that have 1 move left. It is easy to check that $L(M_R) = (\{a\} \oplus \{a\}^+) \cup ((\{a\} \oplus \{a\}^+) \ominus (\{b\} \oplus \{b\}^+) \ominus (\{c\} \oplus \{c\}^+) \ominus (\{a\} \oplus \{a\}^+))$.

Left Start: From grammar $G'_{d,L}$ that we have constructed above, let us consider only the reachable rules $Reach-P'_{d,L}$. We follow mirror-image construction given in Theorem 25 and in order to obtain M_L . We construct $G_{d,L}$ that generates $L_{Rect}(G_{d,L})$. $G_{d,L} = (N_{d,L}, \Sigma, P_{d,L}, S'_L, \#)$ where $N_{d,L} = ((\{A, B, C\} \cup \{\Phi\}) \times (\{A, B, C\} \times \{0, 1\} \times \{L\} \cup S) \times \{A, B, C\}) \cup \{S'_L\}$, where $S'_L \notin N$ and $\Phi \notin N$, i. e., $N_{d,L}$ has 85 non-terminals and $P_{d,L}$ has 245 rules, when we consider only the reachable rules that are useful. We now delete unreachable and useless rules and obtain $P_{d,L}^{Reach}$.

$$\begin{aligned}
P_{d,L}^{Reach} &= \{S'_L\# \rightarrow b(C, B, 0, L, S)\} \\
&\cup \left\{ \begin{array}{c} (C,B,0,L,S) \\ \# \end{array} \rightarrow \begin{array}{c} a \\ (A,C,1,L,C) \end{array}, \begin{array}{c} (A,C,1,L,C) \\ \# \end{array} \rightarrow \begin{array}{c} c \\ (\Phi,A,0,L,A) \end{array} \right\} \\
&\cup \{\#(A, C, 1, L, C) \rightarrow (A, C, 1, L, C)c\} \\
&\cup \{(C, B, 0, L, S)\# \rightarrow b(C, B, 0, L, S)\} \\
&\cup \{(\Phi, A, 0, L, A)\# \rightarrow a(\Phi, A, 0, L, A), (\Phi, A, 0, L, A) \rightarrow a\}.
\end{aligned}$$

Now, we can simulate rules from $P_{d,L}^{Reach}$ to obtain BFA M_L as in Fig. 4.14. Observe that

$$L(M_L) = (\{b\}^+ \oplus \{a\}) \ominus \{c\}^+ \ominus \{a\}^+.$$

As the number of columns must coincide in each row, it can be seen that

$$L(M_L) = (\{b\}^+ \oplus \{a\}) \ominus (\{c\} \oplus \{c\}^+) \ominus (\{a\} \oplus \{a\}^+).$$

We rename the start state of M_L as in the final construction of Theorem 25. Now finally, we combine all the three BFAs: M_L , M_R and M_D those we have obtained from each case, to form one BFA M_{LRD} as in Fig. 4.15 which finalizes the illustration.

4.5 Pumping and Interchange Lemmas

Pumping and interchange lemmas are the combinatorial core tool of many non-inclusion results in Formal Languages and are hence of crucial importance to show strictness of inclusions between language families.

4.5.1 Pumping Lemmas

Since in the pictures of an RML, the first row as well as the columns are generated by regular grammars, there are two ways to apply the pumping lemma for regular languages: we can pump the first row, which results in repetitions of a column-factor of the picture, or we can pump each column individually, which will only lead to a rectangular shaped picture if the pumping exponents are, in a sense, well-chosen. Hence, we can conclude a horizontal and a vertical pumping lemma for RML (see [72]) and, due to Theorem 24, these pumping lemmas carry over to BFA languages in the following way.

Lemma 33. [30] *Let M be a BFA. Then there exists an $n \in \mathbb{N}$, such that, for every $W \in L(M)$ with $|W|_r \geq n$, $W = X \ominus Y \ominus Z$, $|X \ominus Y|_r \leq n$, $|Y|_r \geq 1$ and, for every $k \geq 0$, $X \ominus Y_k \ominus Z \in L(M)$.*

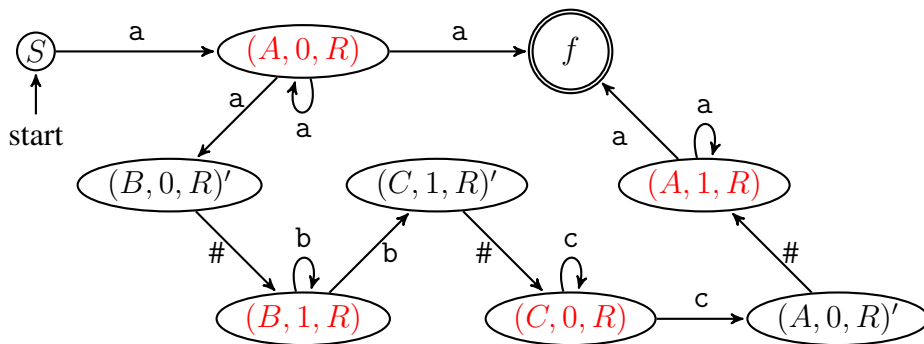


Figure 4.13: BFA M_R that accepts the language $L_{Rect}(G_{d,R})$ in Illustration 8

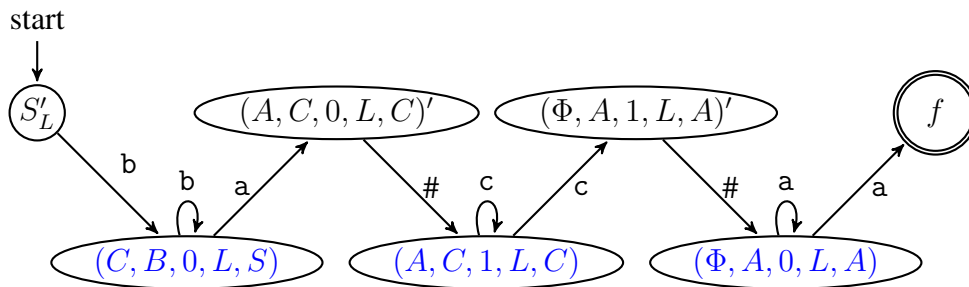


Figure 4.14: BFA M_L that accepts the language $L_{Rect}(G_{d,L})$ in Illustration 8

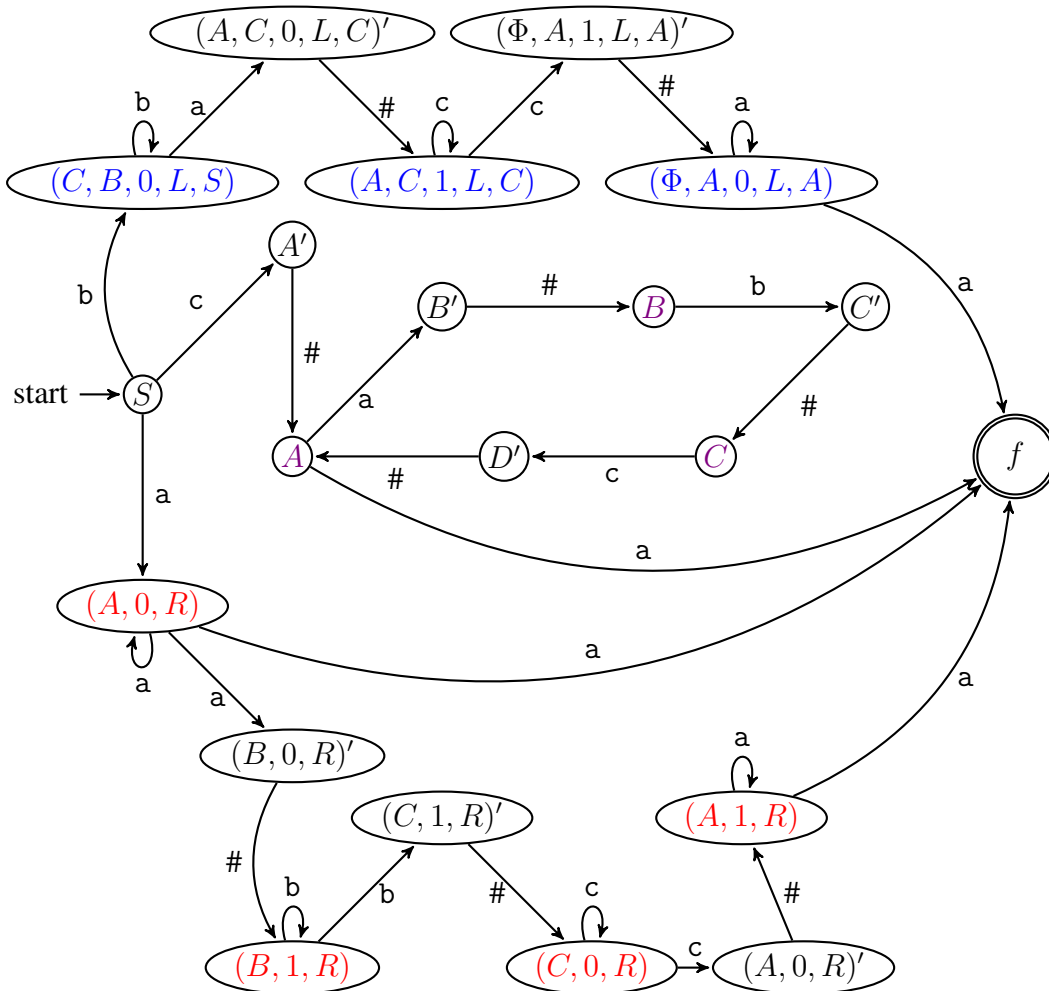


Figure 4.15: BFA M_{LRD} that accepts the language $L_{Rect}(G_d)$ in Illustration 8

Lemma 34. [31] *Let M be a BFA. Then there exists an $n \in \mathbb{N}$, such that, for every*

$$(x_1 \oplus y_1 \oplus z_1) \ominus (x_2 \oplus y_2 \oplus z_2) \ominus \dots \ominus (x_m \oplus y_m \oplus z_m) \in L(M),$$

with $|y_i|_c \geq n$, $1 \leq i \leq m$, there exist factorisations $y_i = u_i \oplus v_i \oplus w_i$ with $|v_i|_c \geq 1$, $1 \leq i \leq m$, such that, for every $k_1, k_2, \dots, k_m \in \mathbb{N}$ with $|v_1|(k_1 - 1) = |v_2|(k_2 - 1) = \dots = |v_m|(k_m - 1)$,

$$(x_1 \oplus u_1 \oplus v_1^{k_1} \oplus w_1 \oplus z_1) \ominus \dots \ominus (x_m \oplus u_m \oplus v_m^{k_m} \oplus w_m \oplus z_m) \in L(M).$$

Lemma 33 is straightforward and to see that Lemma 34 holds, it is sufficient to note that n is the maximum of all pumping lemma constants for the individual rows (recall that each row is generated by an individual regular grammar), so in each row any factor of length at least n contains a factor that can be pumped. Obviously, not every way of pumping the rows results in a rectangular shaped picture, so the pumping exponents must be restricted accordingly.

While the vertical pumping lemma has the nice property that a whole row-factor can be pumped, in the horizontal pumping lemma we can only pump the factors of each individual row, that are independent from each other. As a result, this lemma does not guarantee the possibility of pumping by 0, i. e., removing a factor, which, for classical regular languages, often constitutes a particularly elegant way of showing the non-regularity of a language.

However, it was shown in [30] that also for BFA there exists a horizontal pumping lemma that pumps whole column-factors (which then also translates into a vertical pumping lemma for RML that pumps whole row-factors).

Lemma 35. [30] *Let M be a BFA and let $m \in \mathbb{N}$. Then there exists an $n \in \mathbb{N}$, such that, for every $W \in L(M)$ with $|W|_r \leq m$ and $|W|_c \geq n$, $W = X \oplus Y \oplus Z$, $|X \oplus Y|_c \leq n$, $|Y|_c \geq 1$ and, for every $k \geq 0$, $X \oplus Y^k \oplus Z \in L(M)$.*

4.5.2 Interchange Lemmas

We wish to point out that in a similar way, we can also prove a row and a column interchange lemma (the only difference is that the number n has to be chosen large enough to enforce repeating pairs of states):

Lemma 36. [30] *Let M be a BFA. Then there exists an $n \in \mathbb{N}$, such that, for every $W \in L(M)$ with $|W|_r \geq n$, there exists a factorisation $W = V_1 \ominus X \ominus V_2 \ominus Y \ominus V_3$, $|X|_c \geq 1$, $|Y|_c \geq 1$, such that $V_1 \ominus Y \ominus V_2 \ominus X \ominus V_3 \in L(M)$.*

Lemma 37. [30] *Let M be a BFA and let $m \in \mathbb{N}$. Then there exists an $n \in \mathbb{N}$, such that, for every $W \in L(M)$ with $|W|_r \leq m$ and $|W|_c \geq n$, there exists a factorisation $W = V_1 \oplus X \oplus V_2 \oplus Y \oplus V_3$, $|X|_c \geq 1$, $|Y|_c \geq 1$, such that $V_1 \oplus Y \oplus V_2 \oplus X \oplus V_3 \in L(M)$.*

4.5.3 Application of Pumping and Interchange Lemmas

This section shows several applications of the previous lemmas that are useful for the hierarchy results that we are going to provide below. Also, these lemmas help to understand the limitations of this approach to areas like Character Recognition.

We start with a more illustrative example for Lemma 36. To this end, recall L_{\setminus} from Example 22, the set of pictures of diagonal lines from the upper-left corner to the lower-right corner (realised with some binary alphabet), i. e.,

$$L_{\setminus} = \left\{ \begin{array}{c} \square, \square, \square, \square, \square, \dots \end{array} \right\}.$$

If L can be recognised by a BFA, then, according to Lemma 36, there is a picture $W \in L$ with $W = V_1 \oplus X \oplus V_2 \oplus Y \oplus V_3$, $|X|_c \geq 1$, $|Y|_c \geq 1$, and $W' = V_1 \oplus Y \oplus V_2 \oplus X \oplus V_3 \in L(M)$. The following illustrates how this leads to a contradiction:

$$W = \begin{array}{c} \square \\ = \\ \begin{array}{c} V_1 \\ X \\ V_2 \\ Y \\ V_3 \end{array} \begin{array}{c} \square \\ \hline \square \\ \hline \square \\ \hline \square \\ \hline \square \end{array} \end{array}, \quad W' = \begin{array}{c} \begin{array}{c} V_1 \\ Y \\ V_2 \\ X \\ V_3 \end{array} \begin{array}{c} \square \\ \hline \square \\ \hline \square \\ \hline \square \\ \hline \square \end{array} \end{array}.$$

We chose L to only contain square pictures with a diagonal connecting the upper-left corner with the lower-right one for presentational reasons. In the same way, it can be shown that the set of single continuous diagonal lines cannot be recognised by BFA. Similarly, by Lemma 37, or Pumping Lemmas 33 and 35 in order to show that L cannot be recognised by a BFA.

Next, we show that the set of pictures containing only vertical stripes cannot be accepted by a BFA.

Lemma 38. *The language $L_{|}$ with a vertical line is not accepted by any BFA.*

Proof. As in Example 2, $L_{|}$ is a picture language over the alphabet $\{0, 1\}$, where the vertical line consists of occurrences of the symbol 1. Let us assume that $L_{|}$ can

be recognised by a BFA M and let $W = (0^n \oplus 1 \oplus 0^n) \ominus (0^n \oplus 1 \oplus 0^n) \in L_{\perp}$, where n is the number of Lemma 34. According to Lemma 34, there are factorisations $0^n = u_1 \oplus v_1 \oplus w_1 = u_2 \oplus v_2 \oplus w_2$ with $|v_i|_c \geq 1$, $i \in \{1, 2\}$, such that $W' = (u_1 \oplus (v_1)^{|v_2|+1} \oplus w_1 \oplus 1 \oplus 0^n) \ominus (0^n \oplus 1 \oplus u_2 \oplus (v_2)^{|v_1|+1} \oplus w_2) \in L_{\perp}$ (note that since $|v_1|((|v_2|+1)-1) = |v_2|((|v_1|+1)-1)$, these pumping exponents lead to a rectangular picture). However, since $|u_1 \oplus (v_1)^{|v_2|+1} \oplus w_1|_c > n$, $W' \notin L_{\perp}$, which is a contradiction. \square

Lemma 39. *The language L_{-} with a horizontal line cannot be generated by any RMG.*

Proof. By Example 4, $T(L_{-}) = L_{\perp}$. So, if L_{-} would be generated by some RMG, then $T(L_{-})$ would be accepted by some BFA according to Theorem 24, contradicting Lemma 38. \square

Recall the language L_{\setminus} from Example 22.

Lemma 40. *The language L_{\setminus} of square pictures with ones on the main diagonal and zeros elsewhere is not accepted by any BFA, nor generated by any RMG.*

Proof. For BFAs, this has been shown at the beginning of this section. Since $T(L_{\setminus}) = L_{\setminus}$, if L_{\setminus} would be generated by some RMG, then $T(L_{\setminus})$ would be accepted by some BFA according to Theorem 24, a contradiction. \square

Recall the language L_1 from Example 2.

Lemma 41. *The language $(L_1)^+$ of vertical stripes is not in $\mathcal{L}_{\{0,1\}}(\text{BFA})$.*

Proof. We can easily adapt the proof of Lemma 39. Now, we have to discuss the array $W = (1 \oplus 0^n \oplus 1 \oplus 0^n) \ominus (1 \oplus 0^n \oplus 1 \oplus 0^n) \in (L_1)^+$, where n is the number of Lemma 34. The remaining details are left to the reader. \square

4.6 Hierarchy Results, Further Automata Models

In this section, we review three types of language hierarchy results.

4.6.1 BFA Languages and Regular Matrix Languages

We start with a summary of results, including some application of combinatorial lemmas of the preceding section.

Theorem 26. *For each Σ with $|\Sigma| > 1$, $\mathcal{L}_{\Sigma}(\text{BFA}) \cup \mathcal{L}_{\Sigma}(\text{RMG}) \subsetneq \mathcal{L}_{\text{Rect}, \Sigma}(\text{IRAG})$.*

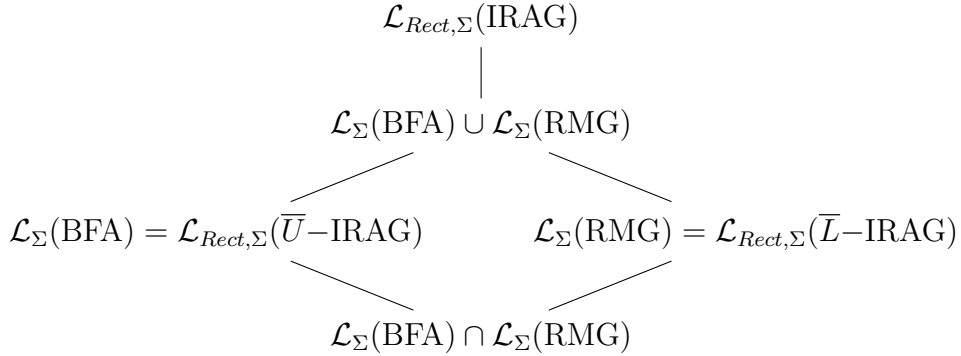


Figure 4.16: Relations between array language families if $|\Sigma| > 1$.

Proof. The inclusion is a consequence of Theorem 25 and Corollary 12. The strictness of the inclusion can be seen by the array language L_{\setminus} from Example 22 and Lemma 40. \square

Figure 4.16 shows inclusion diagram for non-unary array languages, displaying strict inclusions and incomparabilities, also compare the Lemmas 38 and 39 and Examples 15 and 19. For the ease of reference, now we explicitly mention the consequence of this reasoning:

Lemma 42. *Let $|\Sigma| > 1$. Neither $\mathcal{L}_{\Sigma}(\text{BFA})$ nor $\mathcal{L}_{\Sigma}(\text{RMG})$ is closed under T .*

For the unary case, we know that all the language families coincide except for $\mathcal{L}_{Rect,\{a\}}(\text{IRAG})$, for which we only **conjecture** equality with $\mathcal{L}_{\{a\}}(\text{BFA})$ as we did in [34]. We summarize these observations as follows.

Corollary 13. *For non-unary alphabets Σ , the inclusion diagram displayed in Figure 4.16 is correct.*

A special case of picture languages are formed by those over a one-letter alphabet, where (consequently) only shapes prevail. In this case, we find the following:

Proposition 27. [31] *If $|\Sigma| = 1$, then $\mathcal{L}_{\Sigma}(\text{BFA}) = \mathcal{L}_{\Sigma}(\text{RMG})$.*

Corollary 14. $\mathcal{L}_{\{a\}}(\text{BFA}) \cap \mathcal{L}_{\{a\}}(\text{RMG}) = \mathcal{L}_{\{a\}}(\text{BFA}) \cup \mathcal{L}_{\{a\}}(\text{RMG})$.

This contrasts with the conjecture formulated after Theorem 26, which is the only open question regarding the relations between the language families mentioned in Fig. 4.16 for the unary case. It might be also helpful to revisit the considerations of Anselmo, Giammarresi and Madonia [3] on regular expressions for unary array languages here.

4.6.2 3-Way Automata

Yet another interesting related class of non-isometric array languages is the one accepted by 3-way two dimensional DFA [54, 65, 55, 93], denoted as 3-DFA and 3-NFA. We do not give the formal definition of the automaton here, as we are not really making use of it. However, let us list some facts that are interesting when comparing them with our model:

- These automata are quite similar to BFA in the way they scan a picture:
 - They move their head only left, right and down across the picture, but they cannot move upwards. However, they can scan symbols multiple times, as they can freely move left and right.
 - They can also only sense but not move onto the left and right border symbols. However, in accordance with the previous item, they can also move, e. g., left when sensing the rightmost border.
 - In addition, they can also sense the lower border (which we did not introduce for BFA).
- $\mathcal{L}(3 - \text{DFA}) \subsetneq \mathcal{L}(3 - \text{NFA})$; see [52].
- As these automata are similar to BFA in the way they scan the pictures, if we restrict 3-DFA and 3-NFA to move down only at the end, then these 3-way models are equivalent to BFA. To see this, compare with the (now) classical constructions showing the equivalence of one-way and two-way automata models in the case of string languages [113].

Let us express the most important relation to our new model in the following statement.

Theorem 28. $\mathcal{L}(\text{BFA}) \subsetneq \mathcal{L}(3 - \text{NFA})$.

Proof. It is clear from above description that 3-way DFAs can simulate BDFAs. The languages L_{\setminus} and $L_{|}$, known from previous examples, can all be accepted by appropriate 3-way DFAs. This shows strictness of the inclusions. We only sketch how to accept the diagonal language L_{\setminus} , using left-right movements and down movements, starting in the leftmost upper corner of the array.

1. Check if the first row contains only one occurrence of 1 (using left and right movements), namely in its leftmost corner.
2. After the check, move back to the unique occurrence of 1 in this row.

3. Then, move one square to the right and one square down and check if this square contains a 1. If so, check if in this row does not occur any other occurrences of 1. If this check is passed, goto Step 2.
4. If moving right in Step 3 is no longer possible, check if moving down is not possible, either. If both moves are not possible, then we encountered a 1 in the rightmost lower corner of the array, and the automaton accepts the input array.

Observe how we used reading one position multiple times. □

From [53, Theorem 6.4] and also from [57], it follows that 3-NFA array languages are not closed under quarter-turn. However, in order to compare RMGs with 3-NFAs, we now present another rather simple example that also shows this fact.

Proposition 29. [31] $\mathcal{L}(\text{RMG}) \setminus \mathcal{L}(3\text{-DFA}) \neq \emptyset$.

Our previous considerations also show that $\mathcal{L}(\text{RMG})$ is a proper subclass of 3-DFAs with rotated inputs as considered in [57]. Let us only mention that 3-way finite automata can be further generalized to 4-way automata, see [41], and these automata can again be simulated by yet another model, so-called OTAs.

These characterize a more general form of regular array languages, the so-called recognizable picture languages.

Let us finally remark that previously 3-way automata have also been studied when fed with rotated inputs. As (classical) 3-way DFAs can easily simulate BFAs, we get as a corollary to our previous considerations that RMGs can be simulated by 3-way deterministic finite automata with rotated inputs, see [56, 57].

4.6.3 Isometric Array Languages

Conversely, in particular as BFAs process pictures rather in an isometric way, we can use them also to describe non-rectangular arrays. Informally speaking, the processing should start in the uppermost row that contains a symbol from Σ . In this row, the processing starts on the leftmost symbol, moves to the right, until it reaches background symbol $\#$. Upon sensing it, automaton moves downwards to the next row and continues processing it by moving to the left, until again a background symbol $\#$ is sensed, when the next row is processed, etc.

For comparing these isometric array language classes, it is better to go for looking at equivalence classes of languages under translation, usually denoted by square brackets. So, we arrive at classes like $[\mathcal{L}_{\Sigma}^{iso}(\text{BFA})]$ or $[\mathcal{L}_{\Sigma}(\text{IRAG})]$. The

picture of the language families become more intricate here. An illustration can be found in Figure 4.17.

Theorem 30. *For isometric array language classes, inclusion, incomparability relations stated in Figure 4.17 holds for any alphabet Σ with $|\Sigma| > 1$.*

Proof. Basically, all arguments can be borrowed from the non-isometric case. Notice, however, that the (unary!) picture $\begin{smallmatrix} a \\ a \\ a \end{smallmatrix}$ cannot be described by any BFA-type mechanism, while three directions of movements suffice. \square

As an interesting side note, let us mention that

$$[\mathcal{L}_\Sigma(\overline{U}\text{-IRAG})] = [\mathcal{L}_\Sigma(\overline{D}\text{-IRAG})]$$

as well as

$$[\mathcal{L}_\Sigma(\overline{R}\text{-IRAG})] = [\mathcal{L}_\Sigma(\overline{L}\text{-IRAG})]$$

can be seen by the ‘mirror-image construction’ performed above on the level of RMG.

4.7 Closure Properties

We are now ready to study closure properties of the language families that we are interested in. As often, it will be useful that we obtained several characterizations of our language families already. As a by-product, we will obtain that many of the previously introduced language families coincide.

4.7.1 Set Operations

$\mathcal{L}(\text{RMG})$ are closed under Boolean operations. More precisely, it was shown in [106] that $\mathcal{L}(\text{RMG})$ (hence $\mathcal{L}(\text{BFA})$, since $T(L_1 \cup L_2) = T(L_1) \cup T(L_2)$) are closed under union, using a standard grammar construction. We supplement this by the following two results.

Theorem 31. *For each alphabet Σ , $\mathcal{L}_\Sigma(\text{BFA})$ is closed under complementation.*

Proof. First, let us recall from Theorem 22 that BDFA and BFA describe the same class of picture languages. Let $M = (Q, \Sigma, R, s, F, \#, \square)$ be some BDFA. Then we can construct a BDFA \overline{M} by state complementation, i. e., $\overline{M} = (Q, \Sigma, R, s, Q - F, \#, \square)$. On some input picture $A \in \Sigma_+^+$, M reaches the same state as \overline{M} and, furthermore, since both M and \overline{M} are deterministic, there exists exactly one state $q \in Q$ that can be reached by M and \overline{M} on input A . This directly implies that $A \in L(M)$ if and only if $A \notin L(\overline{M})$; thus, $L(\overline{M}) = \overline{L(M)}$. Hence BFA picture languages are closed under complementation. \square

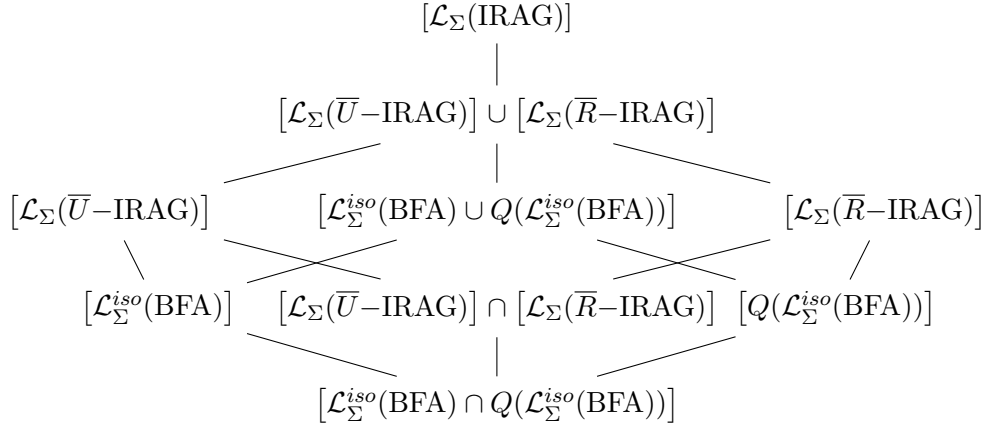


Figure 4.17: Relations between isometric array language families

Notice that the previous theorem has become easy because we have a deterministic model for BFAs, in contrast to what has been established for RML before. De Morgan's law now immediately yields:

Corollary 15. *For each alphabet Σ , $\mathcal{L}_\Sigma(\text{BFA})$ is closed under intersection.*

4.7.2 Reflection-like Operations

As often, the ease or complication of positive closure result depends on a chosen model. As the three reflection-like operations R_v , R_h and T that we study in this section have similarities to the mirror operation in classical Formal Languages, we try to borrow the according construction, showing that the regular string languages are closed under reversal (or mirror image).

Theorem 32. [105] $\mathcal{L}_\Sigma(\text{RMG}) = R_v(\mathcal{L}_\Sigma(\text{RMG}))$.

Corollary 16. $\mathcal{L}_\Sigma(\text{BFA}) = R_h(\mathcal{L}_\Sigma(\text{BFA}))$.

Proof. Observe that $R_h = T \circ R_v \circ T$ and apply Theorems 24 and 32. □

Corollary 17. $\mathcal{L}_\Sigma(\text{RMG}) = R_h(\mathcal{L}_\Sigma(\text{RMG}))$.

Proof. By Cor. 11 and 12, the claim follows. □

Corollary 18. $\mathcal{L}_\Sigma(\text{BFA}) = R_v(\mathcal{L}_\Sigma(\text{BFA}))$.

Proof. Observe that $R_v = T \circ R_h \circ T$ and apply Theorem 24 and Cor. 17. □

Notice that we can also deduce the following results for IRAG, proving that the four possibilities of defining 3-way IRAGs processing rectangular arrays only lead to two classes of languages that we also characterized in different ways in this chapter.

Corollary 19. *Let Σ be some alphabet. Then,*

$$\begin{aligned}\mathcal{L}_{Rect,\Sigma}(\overline{R}\text{-IRAG}) &= \mathcal{L}_{Rect,\Sigma}(\overline{L}\text{-IRAG}) = \mathcal{L}_{\Sigma}(\text{RMG}); \\ \mathcal{L}_{Rect,\Sigma}(\overline{D}\text{-IRAG}) &= \mathcal{L}_{Rect,\Sigma}(\overline{U}\text{-IRAG}) = \mathcal{L}_{\Sigma}(\text{BFA}).\end{aligned}$$

In this section we would like to mention one can also think of giving a simple proof when we first consider the closure under R_h and R_v of $\mathcal{L}_{\Sigma}(\text{RMG})$ which is also carried to $\mathcal{L}_{\Sigma}(\text{BFA})$ via Theorem 24 and Lemma 1 This short proof for Theorem 25 is given by us in [31].

4.7.3 Catenation and Catenation Closure

Row and column catenations and catenation closures correspond to catenation and catenation closure operations in the string case; in fact, positive closure properties can be shown in a similar way. In the following proofs, we use the RFA model.

Theorem 33. $\forall L_1, L_2 \in \mathcal{L}_{\Sigma}(\text{RFA}), L_1 \ominus L_2 \in \mathcal{L}_{\Sigma}(\text{RFA})$.

Proof. Let $M_1 = (Q_1, \Sigma, R_1, s_1, F_1, \#, \square)$ and $M_2 = (Q_2, \Sigma, R_2, s_2, F_2, \#, \square)$ be two RFAs. W.l.o.g., $Q_1 \cap Q_2 = \emptyset$. We construct the RFA M^{\ominus} such that $L(M^{\ominus}) = L(M_1) \ominus L(M_2)$. $M^{\ominus} = (Q_{\ominus}, \Sigma, R_{\ominus}, s_1, F_2, \#, \square)$ is defined by $Q_{\ominus} = Q_1 \cup Q_2$, $R_{\ominus} = R_1 \cup R_2 \cup \{f\# \rightarrow s_2 \mid pa \rightarrow f \in R_1, f \in F_1\}$. The idea of the construction is to first process RFA M_1 ; from the final states of M_1 , by reading a $\#$, we connect to the initial state of the second RFA M_2 which is then processed, ending up in the final states of M_2 . \square

Theorem 34. $\forall L \in \mathcal{L}_{\Sigma}(\text{RFA}), L_+ \in \mathcal{L}_{\Sigma}(\text{RFA})$.

Proof. Let $M = (Q, \Sigma, R, s, F, \#, \square)$ be an RFA and let $L = L(M)$. Now, we construct the RFA M_+ that accepts L_+ . $M_+ = (Q, \Sigma, R^+, s, F, \#, \square)$ is defined by $R^+ = R \cup \{f\# \rightarrow s \mid pa \rightarrow f \in R, f \in F\}$, The idea of the construction is to process the RFA M several times. So, once reaching a final state of M , M_+ can restart by reading a $\#$. \square

Theorem 35. *Let Σ be an alphabet with at least two letters. Then, $\mathcal{L}_{\Sigma}(\text{BFA})$ is not closed under column catenation, nor under column catenation plus.*

Proof. Reconsider the languages L_0 and L_1 from Example 2, with $\{0, 1\} \subseteq \Sigma$. It is straightforward to see that both are BFA languages. However, as $L_{\downarrow} = L_0 \oplus L_1$, their column catenation is not a BFA language according to Lemma 38. Moreover, $(L_1)^+ \notin \mathcal{L}_{\Sigma}(\text{BFA})$ due to Lemma 41. \square

Operations	Symbols	References	$\mathcal{L}(\text{BFA})$
Union	\cup	[106]	Yes
Complementation	$-$	Theorem 31	Yes
Intersection	\cap	Corollary 15	Yes
Row Concatenation	\ominus	Theorem 33	Yes
Column Concatenation	\oplus	Theorem 35	No
Row Concatenation Plus	L_+	Theorem 34	Yes
Column Concatenation Plus	L^+	Theorem 35	No
Transpose	T	Lemma 42	No
Reflection about the vertical	R_v	Corollary 18	Yes
Reflection about the horizontal	R_h	Corollary 16	Yes

Table 4.1: Closure properties of the family $\mathcal{L}(\text{BFA})$

Notice that for the counter-example used in the preceding proof, it is important that our alphabet has at least two letters. Namely, as BFA and RML languages coincide on unary alphabets according to Proposition 27, we could even state a positive closure property for unary BFA languages under column catenation.

4.8 Possible Applications to Character Recognition

Character recognition has always been testbed application for picture processing methods. We refer to [26, 27, 1] and the literature quoted therein. In this regard, we are now going to discuss the recognition of some classes of characters, also (sometimes) showing the limitations of our approach, making use of the pumping lemmas that we have shown above (see Section 4.5).

For example, consider the set $L_{L^{\text{Fix}}}$ of all L tokens of all sizes but with fixed proportion, i. e., the ratio between the two arms of L being 1. First three members of $L_{L^{\text{Fix}}}$ are as follows:

$$\begin{array}{c} x \bullet \\ x x \end{array}, \quad \begin{array}{c} x \bullet \bullet \\ x x x \end{array}, \quad \begin{array}{c} x \bullet \bullet \bullet \\ x \bullet \bullet \bullet \\ x \bullet \bullet \bullet \\ x x x x \end{array}.$$

We claim that $L_{L^{\text{Fix}}}$ is not accepted by any BFA. Suppose there exists a BFA to accept $L_{L^{\text{Fix}}}$. Then by Lemma 33 there exists an $n \in \mathbb{N}$, such that, for every $W \in L_{L^{\text{Fix}}}$ with $|W|_r \geq n$, $W = X \ominus Y \ominus Z$, $|X \ominus Y|_r \leq n$, $|Y|_r \geq 1$ and, for every $k \geq 0$, $X \ominus Y_k \ominus Z \in L_{L^{\text{Fix}}}$. But, unfortunately, for many values of k we get L tokens with unequal arms which are not members of $L_{L^{\text{Fix}}}$ which gives a contradiction to our assumption.

On the other hand, as pointed out by Example 16, if we do not require the ratio between the two arms to be fixed, then the corresponding set of pictures can be recognised by a BFA. Similarly, the characters A (if given in the form ⊕), E, F, H, I, P (if given in the form ⊖), or U (if given in the form ⊔) can be recognised by BFA, if we do not require fixed proportions. We will make this explicit in two examples below. In particular, this means that ⊖ , ⊔ , ⊕ , ⊓ , or ⊔ are valid characters, as well. Note that the character I plays a special role: this set of characters can only be recognised by a BFA if it is given in form $\{\cdot_n^{k_1} \oplus x_n \oplus \cdot_n^{k_2} \mid k_1 \leq k, k_2, n \in \mathbb{N}\}$ or $\{\cdot_n^{k_1} \oplus x_n \oplus \cdot_n^{k_2} \mid k_1 \leq k, \text{ or } k_2 \leq k, n \in \mathbb{N}\}$, for some fixed constant $k \in \mathbb{N}$ (i. e., a BFA is not able to recognise the set of all vertical lines, see Lemma 38).

A similar argument applies to the letter T, even if we allow writings like ⊖ . A possible remedy might be to consider the recognition of transposed letters, as well, because horizontal lines can be detected with BFAs, see Example 15.

However, if we insist on fixed proportions, then it can be easily shown that the character classes mentioned above cannot be recognised by BFAs. For example, if the length of an arm of a character (or the distance between two parallel arms) is only allowed to grow in proportion to the length of another arm, then vertical or horizontal pumping lemma shows this class of characters cannot be recognised by a BFA.

More generally, as shown by Lemma 40, even single diagonal lines cannot be detected by BFA, which excludes several classes of characters from the class of BFA languages, e. g., A, K, M, N, X.

Example 24. We define an RFA M^F that accepts the set L_F of tokens F in Fig. 4.18. Accepted samples look as follows:

$\begin{matrix} x & x \\ x & x \\ x & \bullet \end{matrix}, \quad \begin{matrix} x & x \\ x & \bullet \\ x & x \\ x & \bullet \end{matrix}, \text{ or } \quad \begin{matrix} x & x & x \\ x & \bullet & \bullet \\ x & \bullet & \bullet \\ x & x & x \\ x & \bullet & \bullet \end{matrix}, \quad \text{while} \quad \begin{matrix} x & x & x \\ x & \bullet & x \\ x & x & x \\ x & \bullet & \bullet \end{matrix} \quad \text{or} \quad \begin{matrix} x & x & x & x \\ x & \bullet & \bullet & x \\ x & \bullet & \bullet & x \\ x & x & x & x \\ x & x & x & x \\ x & \bullet & \bullet & \bullet \end{matrix}$

are examples from L_P , the language of tokens P.

As an illustration of the construction of Theorem 33, we refer to Fig. 4.20 for an automaton accepting $L_L \oplus L_F$, where L_L is based on Fig. 4.5.

4.9 Possible Applications to Kolam Patterns

Let us formally have the RMG $G = (V_h, V_v, \Sigma_I, \Sigma, S, R^h, R^v)$ that generates the patterns of *aasanapalakai* as described in [105] as follows:

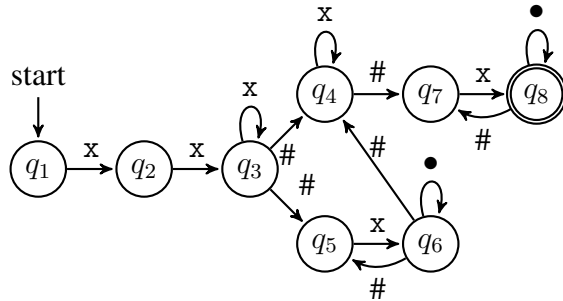


Figure 4.18: RFA M^F that accepts the language of F tokens, of all sizes and of all proportions

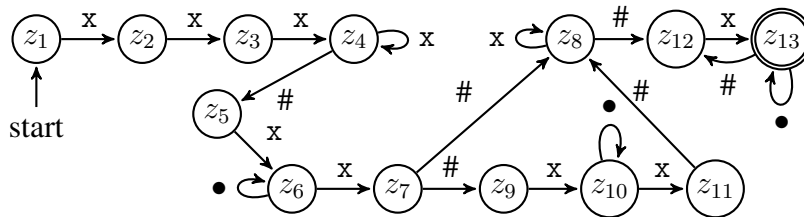


Figure 4.19: RFA M that accepts the language of P tokens, of all sizes and of all proportions

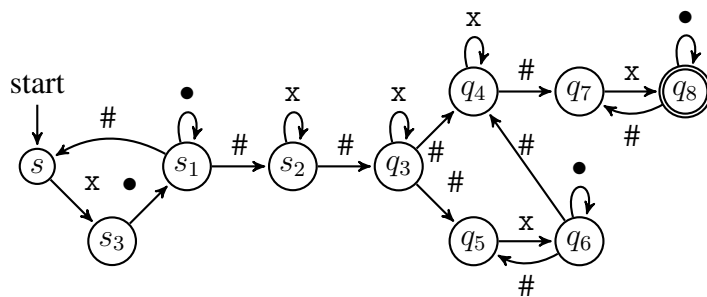
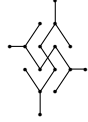





Figure 4.20: RFA accepting $L(M_L) \ominus L(M^F)$

- $V_h = \{S, A, B, C, D\}$;
- $V_v = \{S_1, S_2, S_3, S_4, A_1, B_1, A_2, B_2, C_2, D_2, A_3, B_3, C_3, D_3, A_4, B_4\}$;
- $\Sigma_I = \{S_1, S_2, S_3, S_4\} \subseteq V_v$;
- $\Sigma = \{a, b, c, a_1, a_2, a_3, a_4\}$ where



- a is of the form ,
- b is a blank space,
- c is for the form ,
- a_1 is of the form ,
- $a_2 = R_h(a_1)$, $a_3 = T(a_1)$ and $a_4 = R_h(a_3)$;

- $S \in V_h$ is a *starting symbol*;
- $R^h = \{S \rightarrow S_1C, C \rightarrow S_2B, B \rightarrow S_3C, B \rightarrow S_3A, A \rightarrow S_2D, D \rightarrow S_4\}$;
- $R^v = \{S_1 \rightarrow bA_1, A_1 \rightarrow a_1S_1, A_1 \rightarrow a_1B_1, B_1 \rightarrow b, S_2 \rightarrow a_2A_2, A_2 \rightarrow aB_2, B_2 \rightarrow bA_2, B_2 \rightarrow bC_2, C_2 \rightarrow aD_2, D_2 \rightarrow a_4, S_3 \rightarrow bA_3, A_3 \rightarrow bB_3, B_3 \rightarrow cA_3, B_3 \rightarrow cC_3, C_3 \rightarrow bD_3, D_3 \rightarrow b, S_4 \rightarrow bA_4, A_4 \rightarrow a_3S_4, A_4 \rightarrow a_3B_4, B_4 \rightarrow b\}$.

Now one can apply Theorem 24, to illustrate the kolam *aasanapalakai* via RFA. Instead of illustrating via Theorem 24, we attempt directly to define an RFA

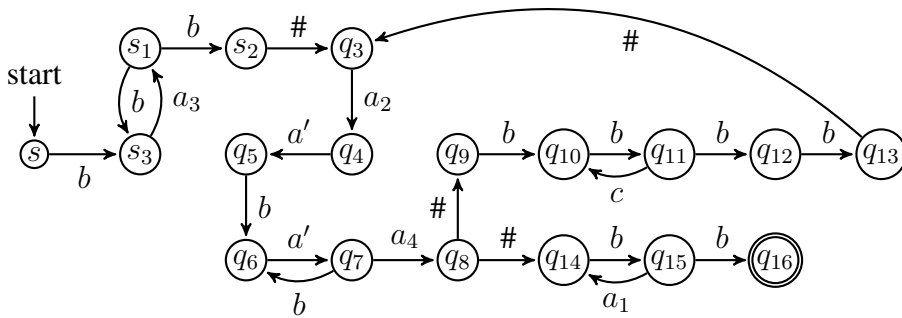


Figure 4.21: RFA accepting transpose of the set of all Aasanapalakai

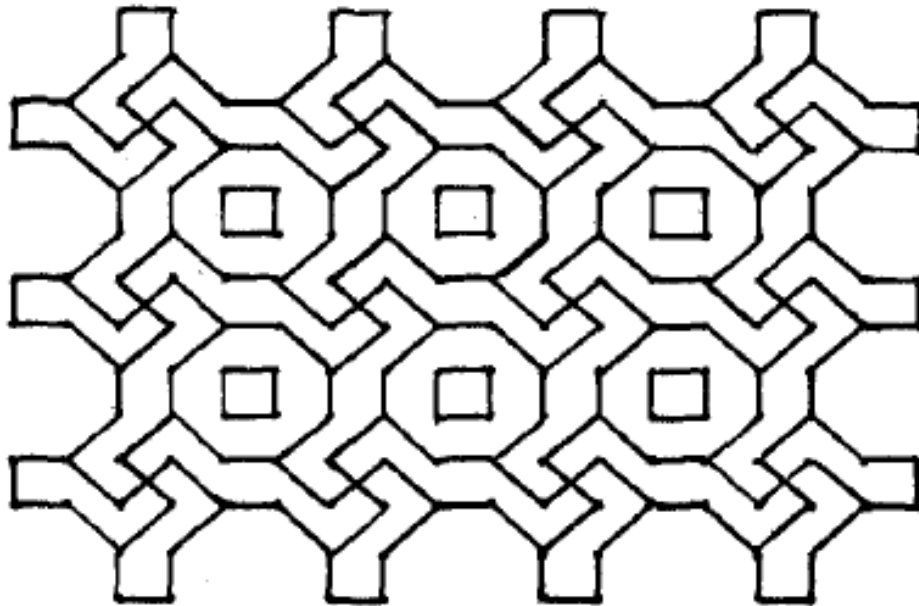


Figure 4.22: A sample element in the set of all Aasanapalakai

considering the transpose of the symbols from Σ especially $a' = T(a)$ as follows:

The smallest picture accepted by RFA in Fig. 4.21 would be

b	a_3	b	a_3	b
a_2	a'	b	a'	a_4
b	b	c	b	b
a_2	a_3	b	a'	a_4
b	a_1	b	a_1	b

this will correspond to an element of the infinite set of pictures $L(G)$, the set of all *aasanapalakai* (a sample is given in Fig. 4.22), for the RMG G which we have define above.

Further examples of this type include *the vine creeper*, *the swing plank* and *the flower cradle* (for figures see [107]).

Chapter 5

Picture Transforming Automata

This chapter can be summarized as follows: (a) We show that variety of picture scanning devices basically only describe two different array language families. (b) This result is obtained by making use of connections to the theory of dihedral groups. (c) Further closure properties of array language families are derived. (d) We also introduce Mealy picture machines and show how they could be useful in a modular design of picture processing automata.

5.1 General Boustrophedon Finite Automata

Now we give one of the main definitions of this section, a new, parameterized automaton model for picture processing [35].

A *general boustrophedon finite automaton*, or GBFA for short, is an 8-tuple $M = (Q, \Sigma, R, s, F, \#, \square, D)$, where Q is a finite set of states, Q is partitioned into Q_f and Q_b , Σ is an input alphabet, $R \subseteq Q \times (\Sigma \cup \{\#\}) \times Q$ is a finite set of rules. A rule $(q, a, p) \in R$ is usually written as $qa \rightarrow p$. We impose some additional restrictions. If $q \in Q_f$ and $a \in \Sigma$, then $qa \rightarrow p \in R$ is only possible if $p \in Q_f$. Such transitions are also called *forward transitions* and collected within R_f . Similarly, if $q \in Q_b$ and $a \in \Sigma$, $qa \rightarrow p \in R$ is only possible if $p \in Q_b$ (*backward transitions*, collected in R_b). Finally, *border transitions* (collected in $R_\#$) are of the form $q\# \rightarrow p$ with $q \in Q_f$ iff $p \in Q_b$. Namely, the special symbol $\# \notin \Sigma$ indicates the border of the rectangular picture that is processed, $s \in Q_f$ is the initial state, F is the set of final states, and $D \in \mathcal{D}$ indicates the move directions. Here,

$$\mathcal{D} = \left\{ \begin{pmatrix} s \rightarrow & \downarrow \\ & \leftarrow \end{pmatrix}, \begin{pmatrix} s \downarrow & \rightarrow \\ & \uparrow \end{pmatrix}, \begin{pmatrix} \downarrow & \leftarrow s \\ \rightarrow & \downarrow \end{pmatrix}, \begin{pmatrix} \leftarrow & \downarrow s \\ \uparrow & \leftarrow \end{pmatrix}, \begin{pmatrix} \rightarrow & \downarrow \\ s \uparrow & \rightarrow \end{pmatrix}, \begin{pmatrix} \uparrow & \leftarrow \\ s \rightarrow & \uparrow \end{pmatrix}, \begin{pmatrix} \downarrow & \leftarrow \\ \leftarrow & \uparrow s \end{pmatrix}, \begin{pmatrix} \rightarrow & \uparrow \\ \uparrow & \leftarrow s \end{pmatrix} \right\}$$

We now discuss notions of configurations, valid configurations, an according

configuration transition to formalize the work of GBFAs, based on snapshots of their work.

Let \square be a new symbol indicating an *erased* position and let $\Sigma_{\#, \square} := \Sigma \cup \{\#, \square\}$. Then $C_M := Q \times (\Sigma_{\#, \square}^{++} \cap (\{\#\}^+ \ominus (\{\#\}_+ \oplus (\Sigma \cup \{\square\})^{++} \oplus \{\#\}_+) \ominus \{\#\}^+)) \times \{f, d\}$ is the set of configurations of M . Hence, the first and last columns and the first and last rows are completely filled with $\#$, and these are the only positions that contain $\#$.

The *initial configuration* is determined by the input array $A \in \Sigma^{++}$. More precisely, if A has m rows and n columns, then

$$(s, \#^{n+2} \ominus (\#_m \oplus A \oplus \#_m) \ominus \#^{n+2}, f)$$

shows the according initial configuration $c_{init}(A)$. Similarly, a *final configuration* is then given by

$$(q_f, \#^{n+2} \ominus (\#_m \oplus \square_m^n \oplus \#_m) \ominus \#^{n+2}, d)$$

for some $q_f \in F$ and $d \in \{b, f\}$.

The processing of the automaton is then crucially depending on $D \in \mathcal{D}$. The arrow that appears together with s indicates the direction of forward processing of the first, third, fifth etc. line. For instance, the first listed direction contains $s \rightarrow$, determining that the odd-numbered rows of the input array are scanned left to right. Similarly, the second listed direction contains $s \downarrow$, determining that the odd-numbered columns of the input array are scanned top to bottom. When the automaton encounters a border symbol, it processes the next line in the reversed way (backward processing).

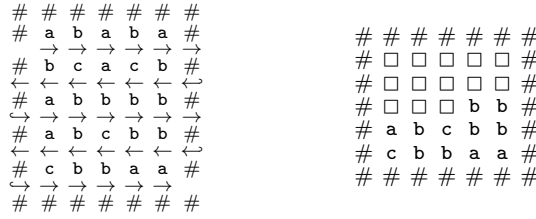
This is also indicated in the little pictures that describe $D \in \mathcal{D}$. For instance, in the first case, the \downarrow in the first row indicates that after hitting the border, the automaton moves downwards, processing (as indicated by the \leftarrow in the last row) now from right to left, until border is hit again, that means to move downwards one more row, as indicated by the \downarrow in the last row. The other $D \in \mathcal{D}$ can be interpreted in a similar fashion, as explained below. Let us now formalize this description. Notice that an odd-numbered row of the input array corresponds to an even-numbered row if we consider the input array bordered by a $\#$ -layer.

- If (p, A, f) and (q, A', f) are two configurations such that A and A' are identical but for one position (i, j) , $1 \leq i \leq m + 2$, $1 \leq j \leq n + 2$, where $A'[i, j] = \square$ while $A[i, j] \in \Sigma$, then $(p, A, f) \vdash_M (q, A', f)$ if $pA[i, j] \rightarrow q \in R_f$. Moreover, i is even.

- Conversely, if (p, A, f) and (q, A', f) are two configurations such that A and A' are identical but for one position (i, j) , $1 \leq i \leq m + 2$, $1 \leq j \leq n + 2$, where $A'[i, j] = \square$ while $A[i, j] \in \Sigma$, then $(p, A, b) \vdash_M (q, A', b)$ if $pA[i, j] \rightarrow q \in R_h$. Moreover, i is odd.
- If (p, A, f) and (q, A, b) are two configurations, then $(p, A, f) \vdash_M (q, A, b)$ or $(p, A, b) \vdash_M (q, A, f)$ if $p\# \rightarrow q \in R_\#$.

The reflexive transitive closure of the relation \vdash_M is denoted by \vdash_M^* . $A \in \Sigma^{++}$ is accepted by a GBFA M with direction $D_{GBFA} := \begin{pmatrix} s \rightarrow & \downarrow \\ \downarrow & \leftarrow \end{pmatrix}$ if $c_{init}(A) \vdash_M^* c$ such that c is a final configuration.

The following illustrates how such a GBFA scans some input picture and how a picture of a valid configuration looks like; it can be seen that the sequence of \square only indicates how far the input has been processed:



It should be also clear that the representation on right-hand side of previous picture contains all information necessary to describe a configuration apart from the state.

Now, we define the other modes by applying transformations according to the following table.

D	$=$	$\begin{pmatrix} s \downarrow & \rightarrow \\ \rightarrow & \uparrow \end{pmatrix}$	$\begin{pmatrix} \downarrow & \leftarrow s \\ \rightarrow & \downarrow \end{pmatrix}$	$\begin{pmatrix} \leftarrow & \downarrow s \\ \uparrow & \leftarrow \end{pmatrix}$	$\begin{pmatrix} \rightarrow & \downarrow \\ s \uparrow & \rightarrow \end{pmatrix}$	$\begin{pmatrix} \uparrow & \leftarrow \\ s \rightarrow & \uparrow \end{pmatrix}$	$\begin{pmatrix} \downarrow & \leftarrow \\ \leftarrow & \uparrow s \end{pmatrix}$	$\begin{pmatrix} \rightarrow & \uparrow \\ \uparrow & \leftarrow s \end{pmatrix}$
$f_D(A)$	$=$	$T(A)$	$R_v(A)$	$Q^{-1}(A)$	$Q(A)$	$R_h(A)$	$T'(A)$	$H(A)$

A is accepted by a GBFA M with a different direction D if $f_D(A)$ is accepted by the GBFA M_{GBFA} that coincides with M in every detail except for the direction, which is now D_{GBFA} . This means, for instance, in the case of $D = \begin{pmatrix} s \downarrow & \rightarrow \\ \rightarrow & \uparrow \end{pmatrix}$, that instead of scanning the input array A column by column, the first column top-down, the second bottom-up, and so forth, we rather transpose A and then scan the transposed array row by row, the first row left-right, the second right-left, and so forth.

The GBFA M is deterministic, or a GB DFA for short, if for each $p \in Q$ and $a \in \Sigma \cup \{\#\}$, there is at most one $q \in Q$ with $pa \rightarrow q \in R$.

This way, we define language classes like $\mathcal{L}_D(\text{GBFA})$ of those array languages accepted by GBFAs working with direction D , as well as

$$\mathcal{L}(\text{GBFA}) := \bigcup_{D \in \mathcal{D}} \mathcal{L}_D(\text{GBFA}).$$

Of course, the interesting question is if the eight language families that we can obtain in these ways are really different from each other or not. Also, the situation of $\mathcal{L}(\text{GBFA})$ needs to be investigated, as well as the role of determinism. From the definitions of GBFAs with their eight working modes and unary operations that we have seen in the group-theoretic excursion from the Subsection 1.2.2, we can immediately derive the following characterization result.

Theorem 36. *The class $\mathcal{L}_{D_{BFA}}(\text{GBFA})$ coincides with the following classes of picture languages: $T \left(\mathcal{L}_{\left(\begin{smallmatrix} s\downarrow & \rightarrow \\ \rightarrow & \uparrow \end{smallmatrix} \right)}(\text{GBFA}) \right)$, $R_v \left(\mathcal{L}_{\left(\begin{smallmatrix} \downarrow & \leftarrow s \\ \rightarrow & \downarrow \end{smallmatrix} \right)}(\text{GBFA}) \right)$, $Q^{-1} \left(\mathcal{L}_{\left(\begin{smallmatrix} \leftarrow & \downarrow s \\ \uparrow & \leftarrow \end{smallmatrix} \right)}(\text{GBFA}) \right)$, $Q \left(\mathcal{L}_{\left(\begin{smallmatrix} \rightarrow & \downarrow \\ s\uparrow & \rightarrow \end{smallmatrix} \right)}(\text{GBFA}) \right)$, $R_h \left(\mathcal{L}_{\left(\begin{smallmatrix} \uparrow & \leftarrow \\ s\rightarrow & \uparrow \end{smallmatrix} \right)}(\text{GBFA}) \right)$, $T' \left(\mathcal{L}_{\left(\begin{smallmatrix} \downarrow & \leftarrow \\ \leftarrow & \uparrow s \end{smallmatrix} \right)}(\text{GBFA}) \right)$, and $H \left(\mathcal{L}_{\left(\begin{smallmatrix} \rightarrow & \uparrow \\ \uparrow & \leftarrow s \end{smallmatrix} \right)}(\text{GBFA}) \right)$.*

Due to the connections to group theory described above, we can easily infer from the previous theorem characterizations of the seven other classes, referring back to $\mathcal{L}_{D_{BFA}}(\text{GBFA})$. We collect these results in the following theorem.

Theorem 37. *We obtain the following list of characterizations.*

- $\mathcal{L}_{\left(\begin{smallmatrix} s\downarrow & \rightarrow \\ \rightarrow & \uparrow \end{smallmatrix} \right)}(\text{GBFA}) = T(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$.
- $\mathcal{L}_{\left(\begin{smallmatrix} \downarrow & \leftarrow s \\ \rightarrow & \downarrow \end{smallmatrix} \right)}(\text{GBFA}) = (Q \circ T)(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$.
- $\mathcal{L}_{\left(\begin{smallmatrix} \rightarrow & \downarrow \\ s\uparrow & \rightarrow \end{smallmatrix} \right)}(\text{GBFA}) = (Q \circ (Q \circ Q))(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$.
- $\mathcal{L}_{\left(\begin{smallmatrix} \leftarrow & \downarrow s \\ \uparrow & \leftarrow \end{smallmatrix} \right)}(\text{GBFA}) = Q(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$.
- $\mathcal{L}_{\left(\begin{smallmatrix} \uparrow & \leftarrow \\ s\rightarrow & \uparrow \end{smallmatrix} \right)}(\text{GBFA}) = (T \circ Q)(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$.
- $\mathcal{L}_{\left(\begin{smallmatrix} \downarrow & \leftarrow \\ \leftarrow & \uparrow s \end{smallmatrix} \right)}(\text{GBFA}) = (Q \circ (Q \circ T))(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$.
- $\mathcal{L}_{\left(\begin{smallmatrix} \rightarrow & \uparrow \\ \uparrow & \leftarrow s \end{smallmatrix} \right)}(\text{GBFA}) = (Q \circ Q)(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$.

These characterizations are also valid for the corresponding deterministic classes.

BFAs that we have seen in the previous chapter basically work as GBFAs do when working in mode D_{BFA} .

Theorem 38. *For each direction mode D , we know: $\mathcal{L}_D(\text{GBFA}) = \mathcal{L}_D(\text{GBDFA})$.*

Proof. By previous reasoning, we can derive from Remark 14 that $\mathcal{L}_D(\text{GBFA}) = \mathcal{L}_D(\text{GBDFA})$ is true for $D = D_{BFA}$. By Theorem 37, we have characterizations of $\mathcal{L}_D(\text{GBFA})$ in terms of $\mathcal{L}_{D_{BFA}}(\text{GBFA})$. These characterizations are also valid for the corresponding deterministic classes. \square

5.2 General Returning Finite Automata

Another model that we have seen was returning finite automata (RFA). We are now going to generalize the work of RFA in the following, again by introducing working modes for them. Now, a pair of directions like $D = (s \rightarrow \downarrow)$ is sufficient, indicating that an input array is always processed row by row, top down, where each row is scanned from left to right; moreover, the procedure is (here) started at the upper left corner of the array, as indicated by the position of s . The processing mode just describes coincide with that of RFAs (See Section 4.2). Leaving out the formal definition for now, we arrive at language families like $\mathcal{L}_D(\text{GRFA})$. It is sufficient to understand that there is no need to give any direction information. There are (again) eight natural processing modes D :

$$(s \rightarrow \downarrow), (s \downarrow \rightarrow), (\downarrow \leftarrow s), (\leftarrow \downarrow s), (s \rightarrow \uparrow), (s \uparrow \rightarrow), (\uparrow \leftarrow s), (\leftarrow \uparrow s).$$

These can be naturally partitioned into the row-first modes $\mathcal{D}_{row-f} = \{(s \rightarrow \downarrow), (\downarrow \leftarrow s), (s \rightarrow \uparrow), (\uparrow \leftarrow s)\}$ and the four other column-first modes in \mathcal{D}_{col-f} . Again, we have deterministic variants, and we can consider the union of all languages pertaining to these GRFA-variants.

Example 25. *The set of all arrays over $\{a, b\}$ such that each array in the set has exactly one row completely filled with b's and a's everywhere else is accepted by a GRFA M as shown in Figure 5.1.*

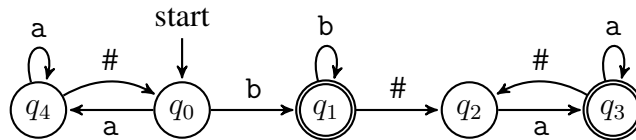


Figure 5.1: GRFA M that accepts the language in Example 25.

As with GBFAs, we can alternatively describe the work of a GRFA working in mode D by first performing a unary operation on the image and then processing the image in the mode $D_{RFA} = (s \rightarrow \downarrow)$ that corresponds to RFAs. Therefore, we obtain the following characterizations for these modes.

Theorem 39. *The class $\mathcal{L}_{D_{RFA}}(\text{GRFA})$ coincides with the following classes:*
 $T(\mathcal{L}_{(s\downarrow \rightarrow)}(\text{GRFA}))$, $R_v(\mathcal{L}_{(\downarrow \leftarrow s)}(\text{GRFA}))$, $Q^{-1}(\mathcal{L}_{(\leftarrow \downarrow s)}(\text{GRFA}))$,
 $Q(\mathcal{L}_{(s\uparrow \rightarrow)}(\text{GRFA}))$, $R_h(\mathcal{L}_{(s \rightarrow \uparrow)}(\text{GRFA}))$, $T'(\mathcal{L}_{(\leftarrow \uparrow s)}(\text{GRFA}))$,
and $H(\mathcal{L}_{(\uparrow \leftarrow s)}(\text{GRFA}))$.

We can use previous theorem to obtain a representation like Theorem 37 also for GRFAs.

Theorem 40. *We obtain the following list of characterizations.*

- $\mathcal{L}_{(s\downarrow \rightarrow)}(\text{GRFA}) = T(\mathcal{L}_{D_{RFA}}(\text{GRFA}))$.
- $\mathcal{L}_{(\downarrow \leftarrow s)}(\text{GRFA}) = (Q \circ T)(\mathcal{L}_{D_{RFA}}(\text{GRFA}))$.
- $\mathcal{L}_{(s\uparrow \rightarrow)}(\text{GRFA}) = (Q \circ (Q \circ Q))(\mathcal{L}_{D_{RFA}}(\text{GRFA}))$.
- $\mathcal{L}_{(\leftarrow \downarrow s)}(\text{GRFA}) = Q(\mathcal{L}_{D_{RFA}}(\text{GRFA}))$.
- $\mathcal{L}_{(s \rightarrow \uparrow)}(\text{GRFA}) = (T \circ Q)(\mathcal{L}_{D_{RFA}}(\text{GRFA}))$.
- $\mathcal{L}_{(\leftarrow \uparrow s)}(\text{GRFA}) = (Q \circ (Q \circ T))(\mathcal{L}_{D_{RFA}}(\text{GRFA}))$.
- $\mathcal{L}_{(\uparrow \leftarrow s)}(\text{GRFA}) = (Q \circ Q)(\mathcal{L}_{D_{RFA}}(\text{GRFA}))$.

We can conclude from Theorem 23:

Corollary 20. $\mathcal{L}_{D_{BFA}}(\text{GBFA}) = \mathcal{L}_{D_{RFA}}(\text{GRFA})$.

By Theorem 37 and Theorem 40, we also get a complete list of correspondences between GBFAs and GRFAs as in Table 5.1. For instance, $Q(\mathcal{L}_{D_{RFA}}(\text{GRFA})) = \mathcal{L}_D(\text{GRFA})$ for $D = (\leftarrow \downarrow s)$ can be read off as the table entry. The previous result means that determinism does not restrict the power of GRFA in all processing modes.

5.3 Language Families under the Unary Operators

We are now going to collect, prove several results relating the different language families that we have introduced so far by means of the unary operators that we discussed above. The proofs will also show that it is quite valuable to have the different processing modes available.

Table 5.1: Operators/Directions for GBFAs and GRFAs.

\mathcal{O}	T	R_v	Q^{-1}	Q	R_h	T'	H
GBFA	$\begin{pmatrix} s \downarrow \rightarrow \\ \rightarrow \uparrow \end{pmatrix}$	$\begin{pmatrix} \downarrow \leftarrow s \\ \rightarrow \downarrow \end{pmatrix}$	$\begin{pmatrix} \rightarrow \downarrow \\ s \uparrow \rightarrow \end{pmatrix}$	$\begin{pmatrix} \leftarrow \downarrow s \\ \uparrow \leftarrow \end{pmatrix}$	$\begin{pmatrix} \uparrow \leftarrow \\ s \rightarrow \uparrow \end{pmatrix}$	$\begin{pmatrix} \downarrow \leftarrow \\ \leftarrow \uparrow s \end{pmatrix}$	$\begin{pmatrix} \rightarrow \uparrow \\ \uparrow \leftarrow s \end{pmatrix}$
GRFA	$\begin{pmatrix} s \downarrow \rightarrow \\ \rightarrow \uparrow \end{pmatrix}$	$\begin{pmatrix} \downarrow \leftarrow s \\ \downarrow \leftarrow s \end{pmatrix}$	$\begin{pmatrix} s \uparrow \rightarrow \\ s \uparrow \rightarrow \end{pmatrix}$	$\begin{pmatrix} \leftarrow \downarrow s \\ \leftarrow \downarrow s \end{pmatrix}$	$\begin{pmatrix} s \rightarrow \uparrow \\ s \rightarrow \uparrow \end{pmatrix}$	$\begin{pmatrix} \leftarrow \uparrow s \\ \leftarrow \uparrow s \end{pmatrix}$	$\begin{pmatrix} \uparrow \leftarrow s \\ \uparrow \leftarrow s \end{pmatrix}$

Lemma 43. $\mathcal{L}_D(\text{GRFA}) = R_v(\mathcal{L}_D(\text{GRFA}))$ for $D \in \mathcal{D}_{\text{row-}f}$.

Proof. Let $M = (Q, \Sigma, R, s, F, \#, \square, (s \rightarrow \downarrow))$ be some GRFA and let $L = L(M)$. Let us construct some GRFA M^v that accepts $R_v(L)$. $M^v = (Q^v, \Sigma, R^v, Q_I, Q_F, \#, \square, (s \rightarrow \downarrow))$ is defined by $Q^v = Q \times Q \times Q$, $Q_\# = \{q \mid pa \rightarrow q \wedge q\# \rightarrow r \in R\}$, $Q_I = \{(s, q, r)\}$ where $q \in Q$, $r \in Q_\#$,

$$\begin{aligned} R^v &= \{(\ell, q, r)a \rightarrow (\ell, p, r) \mid pa \rightarrow q \in R, \ell \in Q, r \in Q_\#, a \in \Sigma\} \\ &\cup \{(\ell, \ell, p)\# \rightarrow (q, t, r) \mid p\# \rightarrow q \in R, \ell, r, t \in Q, a \in \Sigma\} \\ &\cup \{(\ell, q, f)a \rightarrow (\ell, p, f) \mid pa \rightarrow q \in R, \ell \in Q, f \in F, a \in \Sigma\} \end{aligned}$$

and $Q_F \subseteq Q^v$, $Q_F = \{(\ell, \ell, f) \mid f \in F, \ell \in Q\}$.

The idea of the construction is that of the mirror-image construction, well-known from classical formal language theory. Here, the first component ℓ of some triple $(\ell, q, r) \in Q \times Q \times Q$ memorizes the state associated to the left-most symbol of that row, q is the current state and r is associated to the right-most symbol in the row. Reading $\#$ switches to the next row until the final state is reached. We only considered the first processing mode here, the other three modes can be shown using similar constructions. The converse direction follows as $R_v \circ R_v = I$ (See Remark 2). \square

As the vertical reflection can be likewise seen as a change in the processing mode, we can immediately conclude:

Corollary 21. $\mathcal{L}_{(s \rightarrow \downarrow)}(\text{GRFA}) = \mathcal{L}_{(\downarrow \leftarrow s)}(\text{GRFA})$;
 $\mathcal{L}_{(s \rightarrow \uparrow)}(\text{GRFA}) = \mathcal{L}_{(\uparrow \leftarrow s)}(\text{GRFA})$.

Due to Theorem 40, we can also conclude:

Lemma 44. $\mathcal{L}_D(\text{GRFA}) = R_h(\mathcal{L}_D(\text{GRFA}))$ for $D \in \mathcal{D}_{\text{row-}f}$.

Lemma 45. $\mathcal{L}_D(\text{GRFA}) = R_h(\mathcal{L}_D(\text{GRFA}))$ for $D \in \mathcal{D}_{\text{col-}f}$.

Hence, we can immediately conclude the following characterizations.

Corollary 22. $\mathcal{L}_{(s \rightarrow \downarrow)}(\text{GRFA}) = \mathcal{L}_{(s \rightarrow \uparrow)}(\text{GRFA})$;
 $\mathcal{L}_{(\downarrow \leftarrow s)}(\text{GRFA}) = \mathcal{L}_{(\uparrow \leftarrow s)}(\text{GRFA})$.

Corollary 23. $\mathcal{L}_{(s \downarrow \rightarrow)}(\text{GRFA}) = \mathcal{L}_{(s \uparrow \rightarrow)}(\text{GRFA})$;
 $\mathcal{L}_{(\leftarrow \downarrow s)}(\text{GRFA}) = \mathcal{L}_{(\leftarrow \uparrow s)}(\text{GRFA})$.

Lemma 43 and Corollary 20 give the following corollary; also see Theorem 36; similar closure properties for the other processing modes can be easily derived by combining what we have shown so far.

Corollary 24. $\mathcal{L}_{D_{BFA}}(\text{GBFA}) = R_v(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$.

By Corollary 20 and Theorem 39, we obtain:

Theorem 41. $\mathcal{L}_{D_{BFA}}(\text{GBFA}) = Q(\mathcal{L}_{(s \uparrow \rightarrow)}(\text{GRFA}))$.

Theorem 41 immediately yields the following result, as $Q \circ Q^{-1}$ is the identity.

Corollary 25. $Q^{-1}(\mathcal{L}_{D_{BFA}}(\text{GBFA})) = \mathcal{L}_{(s \uparrow \rightarrow)}(\text{GRFA})$.

By Theorems 37, 40, Corollary 20, and Table 5.1, we can conclude the following:

Corollary 26. $\mathcal{L}_{D_{BFA}}(\text{GBFA}) = Q^{-1}(\mathcal{L}_{(s \uparrow \rightarrow)}(\text{GRFA}))$.

As $Q^{-1} \circ Q^{-1} = H$, Corollaries 25 and 26 yield:

Corollary 27. $\mathcal{L}_{D_{BFA}}(\text{GBFA}) = H(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$.

As $R_h = H \circ R_v$, Corollaries 24 and 27 give:

Corollary 28. $\mathcal{L}_{D_{BFA}}(\text{GBFA}) = R_h(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$.

We can now summarize our characterization of picture language classes:

Theorem 42. *The picture language family $\mathcal{L}_{D_{BFA}}(\text{GBFA})$ equals*

- $\mathcal{L}_D(\text{GBFA})$ for $D \in \{D_{BFA}, (\begin{smallmatrix} \downarrow & \leftarrow s \\ \rightarrow & \downarrow \end{smallmatrix}), (\begin{smallmatrix} \uparrow & \leftarrow \\ s \rightarrow & \uparrow \end{smallmatrix}), (\begin{smallmatrix} \rightarrow & \uparrow \\ \uparrow & \leftarrow s \end{smallmatrix})\}$;
- $\mathcal{L}_D(\text{GRFA})$ for $D \in \mathcal{D}_{\text{row-}f}$.

The picture language family $T(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$ equals

- $\mathcal{L}_D(\text{GBFA})$ for $D \in \{(\begin{smallmatrix} s \downarrow & \rightarrow \\ \rightarrow & \uparrow \end{smallmatrix}), (\begin{smallmatrix} \leftarrow & \downarrow s \\ \uparrow & \leftarrow \end{smallmatrix}), (\begin{smallmatrix} \rightarrow & \downarrow \\ s \uparrow & \rightarrow \end{smallmatrix}), (\begin{smallmatrix} \downarrow & \leftarrow \\ \leftarrow & \uparrow s \end{smallmatrix})\}$;
- $\mathcal{L}_D(\text{GRFA})$ for $D \in \mathcal{D}_{\text{col-}f}$.

Proof. (a) $\mathcal{L}_{D_{BFA}}(\text{GBFA}) =_{\text{Cor. 24}} R_v(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{\text{Thm. 37}} \mathcal{L}_{\left(\begin{smallmatrix} \downarrow & \leftarrow s \\ \rightarrow & \downarrow \end{smallmatrix}\right)}(\text{GBFA})$.

$\mathcal{L}_{D_{BFA}}(\text{GBFA}) =_{\text{Cor. 28}} R_h(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{\text{Thm. 37}} \mathcal{L}_{\left(\begin{smallmatrix} \uparrow & \leftarrow \\ s \rightarrow & \uparrow \end{smallmatrix}\right)}(\text{GBFA})$.

$\mathcal{L}_{D_{BFA}}(\text{GBFA}) =_{\text{Cor. 27}} H(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{\text{Thm. 37}} \mathcal{L}_{\left(\begin{smallmatrix} \rightarrow & \uparrow \\ \uparrow & \leftarrow s \end{smallmatrix}\right)}(\text{GBFA})$.

(b) $\mathcal{L}_{D_{BFA}}(\text{GBFA}) =_{\text{Cor. 20}} \mathcal{L}_{(s \rightarrow \downarrow)}(\text{GRFA}) =_{\text{Cor. 21}} \mathcal{L}_{(\downarrow \leftarrow s)}(\text{GRFA})$
 $=_{\text{Cor. 22}} \mathcal{L}_{(\uparrow \leftarrow s)}(\text{GRFA}) =_{\text{Cor. 21}} \mathcal{L}_{(s \rightarrow \uparrow)}(\text{GRFA})$.

(c) $T(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{\text{Thm. 37}} \mathcal{L}_{\left(\begin{smallmatrix} s \downarrow & \rightarrow \\ \rightarrow & \uparrow \end{smallmatrix}\right)}(\text{GBFA})$.

$T(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{\text{Table 1.1a}} Q(R_h(\mathcal{L}_{D_{BFA}}(\text{GBFA})))$
 $=_{\text{Cor. 28}} Q(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{\text{Thm. 37}} \mathcal{L}_{\left(\begin{smallmatrix} \leftarrow & \downarrow s \\ \uparrow & \leftarrow \end{smallmatrix}\right)}(\text{GBFA})$.

$T(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{\text{Table 1.1a}} Q^{-1}(R_v(\mathcal{L}_{D_{BFA}}(\text{GBFA})))$
 $=_{\text{Cor. 24}} Q^{-1}(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{\text{Thm. 37}} \mathcal{L}_{\left(\begin{smallmatrix} \rightarrow & \downarrow \\ s \uparrow & \rightarrow \end{smallmatrix}\right)}(\text{GBFA})$.

$T(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{\text{Table 1.1a}} T'(H(\mathcal{L}_{D_{BFA}}(\text{GBFA})))$
 $=_{\text{Cor. 27}} T'(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{\text{Thm. 37}} \mathcal{L}_{\left(\begin{smallmatrix} \downarrow & \leftarrow \\ \leftarrow & \uparrow s \end{smallmatrix}\right)}(\text{GBFA})$.

(d) $T(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{\text{Cor. 20}} T(\mathcal{L}_{D_{RFA}}(\text{GRFA})) =_{\text{Thm. 40}} \mathcal{L}_{(s \downarrow \rightarrow)}(\text{GRFA})$.

$T(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{(c)} Q(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$
 $=_{\text{Cor. 20}} Q(\mathcal{L}_{D_{RFA}}(\text{GRFA})) =_{\text{Thm. 40}} \mathcal{L}_{(\leftarrow \downarrow s)}(\text{GRFA})$.

$T(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{(c)} Q^{-1}(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$
 $=_{\text{Cor. 20}} Q^{-1}(\mathcal{L}_{D_{RFA}}(\text{GRFA})) =_{\text{Thm. 40}} \mathcal{L}_{(s \uparrow \rightarrow)}(\text{GRFA})$.

$T(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{(c)} T'(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$
 $=_{\text{Cor. 20}} T'(\mathcal{L}_{D_{BFA}}(\text{GBFA})) =_{\text{Thm. 40}} \mathcal{L}_{(\leftarrow \uparrow s)}(\text{GRFA})$.

□

Corollary 29. $\mathcal{L}(\text{GBFA}) = \mathcal{L}_{(s \downarrow \rightarrow)}(\text{GRFA}) \cup \mathcal{L}_{(s \rightarrow \downarrow)}(\text{GRFA})$.

Remark 17. *As can be seen by applying the exchange property arguments in Section 4.5, the set of square-sized arrays of odd length whose middle row and middle column contain b's, while all other positions are filled with a's, is not in $\mathcal{L}(\text{GBFA})$.*

Let \mathcal{O} collect the eight unary operators introduced in Subsection 1.2.2.

Corollary 30. $\forall O \in \mathcal{O} : O(\mathcal{L}(\text{GBFA})) = \mathcal{L}(\text{GBFA})$.

To underline the special importance of the two operations Q and T from \mathcal{O} , we can also state:

Corollary 31. $\mathcal{L}(\text{GBFA}) = Q(\mathcal{L}_{D_{\text{BF}A}}(\text{GBFA})) \cup \mathcal{L}_{D_{\text{BF}A}}(\text{GBFA})$ and $\mathcal{L}(\text{GBFA}) = T(\mathcal{L}_{D_{\text{BF}A}}(\text{GBFA})) \cup \mathcal{L}_{D_{\text{BF}A}}(\text{GBFA})$.

Let us now turn to the two binary catenation operations.

Theorem 43. $\forall L_1, L_2 \in \mathcal{L}_{D_{\text{RF}A}}(\text{GRFA}), L_1 \ominus L_2 \in \mathcal{L}_{D_{\text{RF}A}}(\text{GRFA})$.

Proof. Let $M_1 = (Q_1, \Sigma, R_1, s_1, F_1, \#, \square, (s \rightarrow \downarrow))$ and $M_2 = (Q_2, \Sigma, R_2, s_2, F_2, \#, \square, (s \rightarrow \downarrow))$ be two GRFAs, with $L_1 = L(M_1)$ and $L_2 = L(M_2)$. W.l.o.g., assume that $Q_1 \cap Q_2 = \emptyset$. Let us construct the GRFA M^\ominus that accepts $L_1 \ominus L_2$ (i.e., $L(M^\ominus) = L_1 \ominus L_2$). $M^\ominus = (Q_\ominus, \Sigma, R_\ominus, s_1, F_2, \#, \square, (s \rightarrow \downarrow))$ is defined by $Q_\ominus = Q_1 \cup Q_2$, $R_\ominus = R_1 \cup R_2 \cup \{f\# \rightarrow s_2 \mid f \in F_1\}$. The idea of the construction is to first simulate M_1 ; whenever a final state is entered at the end of reading a line, then the simulation may switch to M_2 . \square

Corollary 32. $\forall L_1, L_2 \in Q(\mathcal{L}_{D_{\text{RF}A}}(\text{GRFA})), L_1 \oplus L_2 \in Q(\mathcal{L}_{D_{\text{RF}A}}(\text{GRFA}))$.

Define a mapping $str : \Sigma_2^+ \rightarrow (\Sigma \cup \{\#\})^+$, $W \mapsto w$ where $w = w_1\#w_2$, $|W|_c = n$, $n \geq 2$ and $|W|_r = 2$; moreover, $|w_1| = |w_2| = n$, so that $|w| = 2n + 1$.

Lemma 46. *If $L \in \mathcal{L}_{D_{\text{RF}A}}(\text{GRFA})$ with $L \subseteq \Sigma_2^+$, then $str(L)$ is context-free.*

Proof. A pushdown automaton uses its finite control as the GRFA and simply checks if the two rows have equal length by using its pushdown store. \square

Theorem 44. $\exists L_1, L_2 \in \mathcal{L}_{D_{\text{RF}A}}(\text{GRFA}) : L_1 \oplus L_2 \notin \mathcal{L}_{D_{\text{RF}A}}(\text{GRFA})$.

Proof. Let $L = \left\{ \begin{smallmatrix} 1 & 0^\ell \\ 1 & 0^\ell \end{smallmatrix} : \ell \geq 1 \right\}$. Clearly, $L \in \mathcal{L}_{D_{\text{RF}A}}(\text{GRFA})$. However, $str(L \oplus L)$ is a variant of the crossing dependency language known to be not context-free. By Lemma 46, $L \oplus L \notin \mathcal{L}_{D_{\text{RF}A}}(\text{GRFA})$. \square

Corollary 33. $\exists L_1, L_2 \in Q(\mathcal{L}_{D_{\text{RF}A}}(\text{GRFA})) : L_1 \ominus L_2 \notin Q(\mathcal{L}_{D_{\text{RF}A}}(\text{GRFA}))$.

Theorem 45. $\mathcal{L}(\text{GRFA})$ is closed neither under column catenation nor under row catenation.

Proof. Consider L from the proof of Theorem 44. By Theorem 43, $L' = L \ominus \{0\}_+^+ \ominus \{1\}_+^+ \ominus \{0\}_+^+ \in \mathcal{L}_{D_{\text{RF}A}}(\text{GRFA})$. If $L' \oplus L' \in \mathcal{L}_{D_{\text{RF}A}}(\text{GRFA})$, then also $L \oplus L \in \mathcal{L}_{D_{\text{RF}A}}(\text{GRFA})$, contradicting the reasoning from Theorem 44. Exchange property arguments in Section 4.5 show that $\{L', L' \oplus L'\} \cap Q(\mathcal{L}_{D_{\text{RF}A}}(\text{GRFA})) = \emptyset$, as horizontal lines in arbitrary positions cannot be checked by finite automata working column by column. \square

So far, we have used finite automata to only accept picture (languages). If such devices should be used in practice, at least two questions show up:

- Can we design such automata in a systematic, best modular fashion?
- Can we set up these automata so that they can tolerate certain input errors?

We will see this in the next section.

5.4 Picture Transforming Automata

To answer both questions, we introduce a generalization of Mealy machines to picture processing. A *Mealy Picture Machine*, or MPM for short, can be specified as a 9-tuple $M = (Q, \Sigma, \Gamma, \delta, \lambda, q_0, \#, F, D)$, where Q is a finite set of states, Σ is a finite set of input symbols, Γ is a finite set of output symbols, $\delta : Q \times (\Sigma \cup \{\#\}) \rightarrow Q$ is the transition function, $\lambda : Q \times \Sigma \rightarrow \Gamma$ is the output function, $q_0 \in Q$ is the initial state, $\# \notin \Sigma$, $\# \notin \Gamma$, is the special symbol that indicates the borders of the picture that is processed, F is the set of final states, and $D \in \mathcal{D}$ indicates the move directions of the GRFAs. Let us now give the notion of configurations to formalize the working of MPMs, based on the snapshots of their work. Here, we assume (w.l.o.g.) that $\Sigma \cap \Gamma = \emptyset$. Let $\Upsilon := \Sigma \cup \Gamma$ and $\Upsilon_{\#} := \Upsilon \cup \{\#\}$. Then $C_M := Q \times (\Upsilon_{\#}^{++} \cap (\{\#\}^+ \oplus (\{\#\}_+ \oplus \Upsilon^{++} \oplus \{\#\}_+)) \oplus \{\#\}^+) \times \mathbb{N}$ is the set of configurations of M . Hence, the first and last columns and the first and last rows are completely filled with $\#$, and these are the only positions that contain $\#$. The *initial configuration* is determined by the input array $A \in \Sigma^{++}$. More precisely, if A has m rows and n columns, then

$$(q_0, \#^{n+2} \ominus (\#_m \oplus A \oplus \#_m) \ominus \#^{n+2}, 1)$$

shows according initial configuration $C_{init}(A)$. Similarly, a *final configuration* $C_{fin}(A')$ is then given by

$$(q_f, \#^{n+2} \ominus (\#_m \oplus A' \oplus \#_m) \ominus \#^{n+2}, n)$$

for some $A' \in \Gamma^{++}$ with m rows and n columns and for some $q_f \in F$. The processing of the machine depends on $D \in \mathcal{D}$ of the GRFAs. Let us now formalize the description with direction $D = (s \rightarrow \downarrow)$; the remaining seven directions can be formalized similarly but we did not explicitly write those here.¹

- If (p, A, μ) and (q, B, μ) are two configurations such that A and B are identical but for one position (i, j) , $1 \leq i \leq m + 2$, $1 \leq j \leq n + 2$, where $B[i, j] \in \Gamma$, while $A[i, j] \in \Sigma$, then $(p, A, \mu) \vdash_M (q, B, \mu)$ if $\delta(p, A[i, j]) = q$ and $\lambda(p, A[i, j]) = B[i, j]$.
- If (p, A, μ) and $(q, A, \mu + 1)$ are two configurations, then $(p, A, \mu) \vdash_M (q, A, \mu + 1)$ if $\delta(p, \#) = q$.

The reflexive transitive closure of the relation \vdash_M is denoted by \vdash_M^* . Notice that for each $A \in \Sigma^{++}$ there is at most one $A' \in \Gamma^{++}$ such that $C_{init}(A) \vdash_M^* C_{fin}(A')$. We can hence view M as a partial function $M : \Sigma^{++} \dashrightarrow \Gamma^{++}$.

¹We could also use the unary operations and their inverses to formally describe the processing.

Theorem 46. Given $L \in \mathcal{L}_{D_{RFA}}(\text{GRFA})$, with $L \subseteq \Sigma^{++}$, and an MPM $M: \Sigma^{++} \rightarrow \Gamma^{++}$, then $M(L) \in \mathcal{L}_{D_{RFA}}(\text{GRFA})$.

Proof. Let $R = (Q, \Sigma, P, s, F, \#, \square, (s \rightarrow \downarrow))$ be some GRFA then $L(R) \subseteq \Sigma^{++}$. Let $M: \Sigma^{++} \rightarrow \Gamma^{++}$ be an MPM, $M = (Q_M, \Sigma, \Gamma, \delta, \lambda, s_M, \#, F_M, (s \rightarrow \downarrow))$ lifted to $M: 2^{\Sigma^{++}} \rightarrow 2^{\Gamma^{++}}$, i.e., $M(\Sigma^{++}) \subseteq \Gamma^{++}$. Now our aim is to define a GRFA R' such that $L(R') = M(L(R))$. $R' = (Q', \Gamma, P', s', F', \#, \square, (s \rightarrow \downarrow))$ is defined by $Q' = Q \times Q_M$, $P' = \{(p, q)c \rightarrow (p', q') \mid \exists a \in \Sigma : pa \rightarrow p' \in P, \delta(q, a) = q' \wedge \lambda(q, a) = c\} \cup \{(p, q)\# \rightarrow (p', q') \mid p\# \rightarrow p' \in P, \delta(q, \#) = q'\}$, $s' = (s, s_M) \in Q'$ is the start state and $F' = F \times F_M$ is the set of final states.

The idea of the construction is that the automaton R' , upon reading $c \in \Gamma$, guesses which symbol $a \in \Sigma$ was translated into c by M , and this guess is verified by a parallel simulation of the work of M and of R on input symbol a . Similarly, the simulation is performed when encountering a border symbol $\#$. An array $A' \in \Gamma^{++}$ is accepted iff the array $A \in \Sigma^{++}$ that is guessed letter-by-letter in the described fashion is accepted by R and translated by M into A' . \square

Theorem 47. Given $L \in \mathcal{L}_{D_{RFA}}(\text{GRFA})$, with $L \subseteq \Gamma^{++}$, and an MPM $M: \Sigma^{++} \rightarrow \Gamma^{++}$, then $M^-(L) \in \mathcal{L}_{D_{RFA}}(\text{GRFA})$ with $M^-(L) \subseteq \Sigma^{++}$.

Proof. Let $R = (Q, \Gamma, P, s, F, \#, \square, (s \rightarrow \downarrow))$ be some GRFA with $L(R) \subseteq \Gamma^{++}$. Let $M: \Sigma^{++} \rightarrow \Gamma^{++}$ be an MPM, $M = (Q_M, \Sigma, \Gamma, \delta, \lambda, s_M, \#, F_M, (s \rightarrow \downarrow))$ lifted to $M: 2^{\Sigma^{++}} \rightarrow 2^{\Gamma^{++}}$ and $M^-: 2^{\Sigma^{++}} \rightarrow 2^{\Gamma^{++}}$, i.e., $M^-(\Gamma^{++}) \subseteq \Sigma^{++}$. Now our aim is to define a GRFA R' such that $L(R') = M^-(L(R))$. $R' = (Q', \Sigma, P', s', F', \#, \square, (s \rightarrow \downarrow))$ is defined by $Q' = Q \times Q_M$, $P' = \{(p, q)a \rightarrow (p', q') \mid \delta(q, a) = q' \wedge p\lambda(q, a) \rightarrow p' \in P\} \cup \{(p, q)\# \rightarrow (p', q') \mid \delta(q, \#) = q' \wedge p\# \rightarrow p' \in P\}$, $s' = (s, s_M) \in Q'$ is the start state and $F' = F \times F_M$. \square

Let us now explain the extended power of GRFA by adding MPM with it. Also, we want to describe how to use MPMs and further closure properties to design more complicated GRFAs, starting off from very simple automata. Suppose the task is to design a GRFA R' with $L(R') = L'$, see Table 5.2. How can we obtain such an automaton? We see that the pictures in L' can be decomposed into two subsequent rows of x and one first column of x . It looks easier to design automata for both tasks separately and then combine them later ‘somehow’. This is what MPMs can do. Assume that we have designed an MPM M that takes inputs from Σ^{++} , with $\Sigma = \{x, \bullet\}$, and does the following:

- It converts all \bullet symbols into a if they do not appear in the first column.
- It converts all x symbols into a if they appear in first column, but into b if they appear in any other column.

Assume we have designed M . $M(L')$ can be found in Table 5.2. How can we further process this? Maybe, it would be an idea to design another MPM M' that takes inputs from $\{a, b\}^{++}$ and converts them according to the following:

- It converts a symbols into \bullet but one special to be mentioned later.
- When it first reads a b, this must be in the second column, and this b must be followed by b's only in the current row, which is then converted into a row of \bullet 's only. (If the first row where M' encounters any b's is not of the form ab^+ , M' will not enter an accepting state when further processing the input array, as it has observed an input error.)
- In the next row, it is checked if the first symbol is a a. This will then be converted to x.
- Any further occurrences of b will be converted into x.

Table 5.2: Simplifying array languages with MPMs

$$L' = \left\{ \begin{array}{c} (x (\bullet)^n)_{\ell-1} \\ x \quad (x)^n \\ x \quad (x)^n \\ (x (\bullet)^n)_{m-1} \end{array} : n, m, \ell \geq 1 \right\},$$

$$M(L') = \left\{ \begin{array}{c} (a (a)^n)_{\ell-1} \\ a \quad (b)^n \\ a \quad (b)^n \\ (a (a)^n)_{m-1} \end{array} : n, m, \ell \geq 1 \right\}.$$

As a consequence, we find:

$$M'(M(L')) = \left\{ \begin{array}{c} (\bullet (\bullet)^n)_{\ell-1} \\ \bullet \quad (\bullet)^n \\ x \quad (x)^n \\ (\bullet (\bullet)^n)_{m-1} \end{array} : n, m, \ell \geq 1 \right\} = \left\{ \begin{array}{c} (\bullet (\bullet)^n)_{\ell} \\ x \quad (x)^n \\ (\bullet (\bullet)^n)_{m-1} \end{array} : n, m, \ell \geq 1 \right\}.$$

We can devise a quite similar machine R that then accepts all arrays over $\{\bullet, x\}$ that start with at least one row filled with \bullet , followed by one row of length at least two completely filled with x and then followed by an arbitrary number (possibly zero many) of rows filled up with \bullet . Now, the GRFA R' we are looking for can be obtained by first constructing R^\dagger for accepting $M'^-(L(R))$, based on R and M' , and then constructing R' accepting $M^-(L(R^\dagger))$, based on R^\dagger and M .

The drawback of this design approach could be a certain state explosion. For instance, already the smallest GRFA implementing $L(R)$ has five states and the smallest MPM for M' has six states, which gives us 30 states for R^\dagger , unless we interleave steps that delete useless states or even implement state minimization procedures.

More Details on Our Example for Modular Design

Now we will give the detailed explanation of our example for modular design that we have mentioned above.

The GRFA R we start with is $R = (Q, \Gamma, P, s, F, \#, \square, (s \rightarrow \downarrow))$, where $Q = \{s_1, s_2, s_3, s_4, s_5\}$, $\Gamma = \{x, \bullet\}$, $P = \{s_1 \bullet \rightarrow s_1, s_2 x \rightarrow s_3, s_3 x \rightarrow s_4, s_4 x \rightarrow s_4, s_5 \bullet \rightarrow s_5, s_1 \# \rightarrow s_1, s_5 \# \rightarrow s_5, s_1 \# \rightarrow s_2, s_4 \# \rightarrow s_5\}$, $s = s_1$ and $F = \{s_4, s_5\}$. A pictorial representation of R is in Figure 5.2.

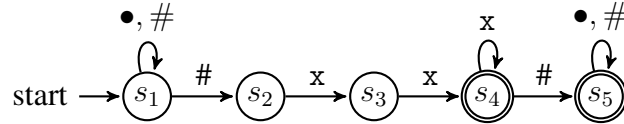


Figure 5.2: GRFA R that accepts $L(R) = \{\bullet\}_+^\dagger \ominus (\{x\} \oplus \{x\}^+) \ominus \{\bullet\}_*^+$.

The MPM $M' = (Q_{M'}, \Sigma, \Gamma, \delta_{M'}, \lambda_{M'}, s_{M'}, \#, F_{M'}, (s \rightarrow \downarrow))$ where $Q_{M'} = \{q_1, \dots, q_6\}$, $\Sigma = \{a, b\}$, $\Gamma = \{x, \bullet\}$, $\delta_{M'}$, $\lambda_{M'}$ are as depicted in Figure 5.3, $s_{M'} = q_1$ and $F_{M'} = \{q_6\}$.

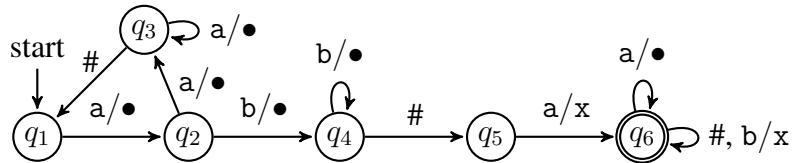


Figure 5.3: MPM M' designed according to our description.

Now let us construct the GRFA R^\dagger that accepts $M'^-(L(R))$, based on R and M' . This construction is an illustration of Theorem 47. The GRFA R^\dagger is given by $R^\dagger = (Q'', \Sigma, P'', s'', F'', \#, \square, (s \rightarrow \downarrow))$ where $Q'' = Q \times Q_{M'}$, i.e., $Q'' = \{s_1, \dots, s_5\} \times \{q_1, \dots, q_6\}$, i.e., Q'' has 30 states, $\Sigma = \{a, b\}$, $s'' = (s_1, q_1)$, $F'' = F \times F_{M'} = \{s_4, s_5\} \times \{q_6\} = \{(s_4, q_6), (s_5, q_6)\}$, and

$$\begin{aligned}
P'' = \{ & (s_1, q_1) \mathbf{a} \rightarrow (s_1, q_2), (s_5, q_1) \mathbf{a} \rightarrow (s_5, q_2), \\
& (s_1, q_2) \mathbf{b} \rightarrow (s_1, q_4), (s_5, q_2) \mathbf{b} \rightarrow (s_5, q_4), \\
& (s_1, q_2) \mathbf{a} \rightarrow (s_1, q_3), (s_5, q_2) \mathbf{a} \rightarrow (s_5, q_3), \\
& (s_1, q_3) \mathbf{a} \rightarrow (s_1, q_3), (s_5, q_3) \mathbf{a} \rightarrow (s_5, q_3), \\
& (s_1, q_4) \mathbf{b} \rightarrow (s_1, q_4), (s_5, q_4) \mathbf{b} \rightarrow (s_5, q_4), \\
& (s_2, q_5) \mathbf{a} \rightarrow (s_3, q_6), (s_3, q_5) \mathbf{a} \rightarrow (s_4, q_6), (s_4, q_5) \mathbf{a} \rightarrow (s_4, q_6) \\
& (s_1, q_6) \mathbf{a} \rightarrow (s_1, q_6), (s_5, q_6) \mathbf{a} \rightarrow (s_5, q_6), \\
& (s_2, q_6) \mathbf{b} \rightarrow (s_3, q_6), (s_3, q_6) \mathbf{b} \rightarrow (s_4, q_6), (s_4, q_6) \mathbf{b} \rightarrow (s_4, q_6) \\
& (s_1, q_3) \# \rightarrow (s_1, q_1), (s_1, q_4) \# \rightarrow (s_1, q_5), (s_1, q_6) \# \rightarrow (s_1, q_6) \\
& (s_1, q_3) \# \rightarrow (s_2, q_1), (s_1, q_4) \# \rightarrow (s_2, q_5), (s_1, q_6) \# \rightarrow (s_2, q_6) \\
& (s_4, q_3) \# \rightarrow (s_5, q_1), (s_4, q_4) \# \rightarrow (s_5, q_5), (s_4, q_6) \# \rightarrow (s_5, q_6) \\
& (s_5, q_3) \# \rightarrow (s_5, q_1), (s_5, q_4) \# \rightarrow (s_5, q_5), (s_5, q_6) \# \rightarrow (s_5, q_6) \}.
\end{aligned}$$

We can note 9 states out of 30 states are not involved in the construction (which means that they directly had fallen into the **not reachable** states) and only 21 states are involved, in these 21 when we ignore the **not reachable** and **useless** states, using the rules from P'' , we arrive at the GRFA R^\dagger with only 8 useful states as depicted in Figure 5.4.

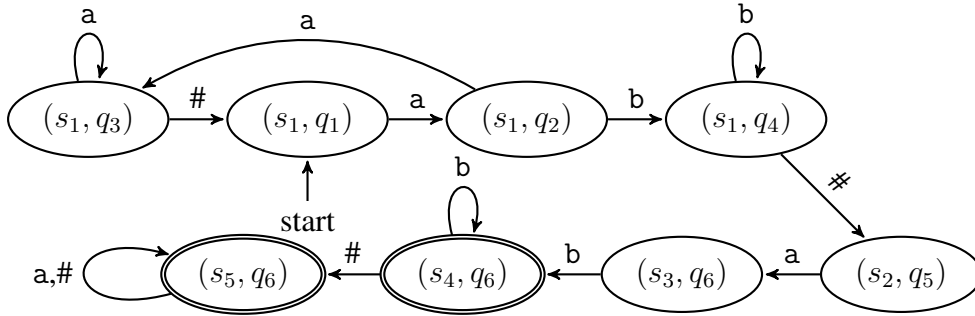


Figure 5.4: GRFA R^\dagger that accepts $M^-(L(R))$

The MPM M is given by $M = (Q_M, \Sigma, \Gamma, \delta_M, \lambda_M, s_M, \#, F_M, (s \rightarrow \downarrow))$ where $Q_M = \{t_1, t_2\}$, $\Sigma = \{\mathbf{x}, \bullet\}$, $\Gamma = \{\mathbf{a}, \mathbf{b}\}$, δ_M, λ_M are as depicted in Figure 5.5, $s_M = t_1$ and $F_M = \{t_2\}$.

Now we construct GRFA R' that we were aiming at, that accepts $M^-(L(R^\dagger))$, based on R^\dagger and M , we can note this construction is an illustration of Theorem 47.

The GRFA $R' = (Q', \Sigma, P', s', F', \#, \square, (s \rightarrow \downarrow))$ where $Q' = Q'' \times Q_M$ i.e., Q' has 60 states, but since we arrived at the GRFA R^\dagger with only 8 useful states as in Figure 5.4, we are interested in only 16 states out of these 60 states,

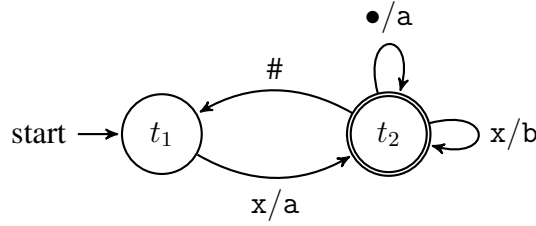


Figure 5.5: MPM M

$\Sigma = \{x, \bullet\}$, $s' = ((s_1, q_1), t_1)$, $F' = F'' \times F_M = \{((s_4, q_6), t_2), ((s_5, q_6), t_2)\}$,
 and $P' = \{((s_1, q_1), t_1)x \rightarrow ((s_1, q_2), t_2), ((s_1, q_2), t_1)x \rightarrow ((s_1, q_3), t_2),$
 $((s_1, q_3), t_1)x \rightarrow ((s_1, q_3), t_2), ((s_2, q_5), t_1)x \rightarrow ((s_3, q_6), t_2),$
 $((s_5, q_6), t_1)x \rightarrow ((s_5, q_6), t_2),$
 $((s_1, q_1), t_2)\bullet \rightarrow ((s_1, q_2), t_2), ((s_1, q_2), t_2)\bullet \rightarrow ((s_1, q_3), t_2),$
 $((s_1, q_3), t_2)\bullet \rightarrow ((s_1, q_3), t_2), ((s_2, q_5), t_2)\bullet \rightarrow ((s_3, q_6), t_2),$
 $((s_5, q_6), t_2)\bullet \rightarrow ((s_5, q_6), t_2),$
 $((s_1, q_2), t_2)x \rightarrow ((s_1, q_4), t_2), ((s_1, q_4), t_2)x \rightarrow ((s_1, q_4), t_2),$
 $((s_3, q_6), t_2)x \rightarrow ((s_4, q_6), t_2), ((s_4, q_6), t_2)x \rightarrow ((s_4, q_6), t_2),$
 $((s_1, q_3), t_2)\# \rightarrow ((s_1, q_1), t_1), ((s_1, q_4), t_2)\# \rightarrow ((s_2, q_5), t_1),$
 $((s_4, q_6), t_2)\# \rightarrow ((s_5, q_6), t_1), ((s_5, q_6), t_2)\# \rightarrow ((s_5, q_6), t_1)\}$,

here we can note that 3 states out of 16 states are not at all involved in the construction (directly fallen into the **not reachable** states) and only 13 states are involved, in these 13 states when we ignore again the **not reachable**, using the rules from P' , we will finally be arriving at the GRFA R' that we have aimed at with 9 useful states as depicted in Figure 5.6.

Please note that here the **useless** states are not in the discussion because they have been already got ignored when we selected only 16 states out of 60 states from Q' .

This example already shows that lazy evaluation approach, aiming to provide states only when they become necessary, is very useful and practical to ease the modular design of picture processing automata.

Notice that MPMs naturally generalize array homomorphisms as considered in [105]. Also, with the possibility to implement multiple passes (by applying the described form of transductions), it is easy to also implement set operations like union or intersection, as the first reading automaton could communicate with the

automaton doing the second read by changing one particular part of the picture, for instance, by printing a special character at the very end, signaling acceptance. However, we also have negative results like the following one.

Theorem 48. $T(\mathcal{L}(\text{GRFA}))$ is not closed under MPM nor under inverse MPM mappings.

Proof. Exchange property argument in Section 4.5, shows that the language $L = \{a\}_+^+ \ominus \{b\}^+ \ominus \{a\}_+^+ \notin T(\mathcal{L}(\text{GRFA}))$. However, it is easy to design an MPM M that only translates arrays from L into some arrays over the alphabet $\{0\}$. Also, $\{0\}_+^+ \in T(\mathcal{L}(\text{GRFA}))$. This shows non-closure under inverse MPM mappings, as $L = M^{-1}(\{0\}_+^+)$. Conversely, let the MPM M' work on arrays over the alphabet $\{0, 1\}$ as follows: If a row starts with 0, then M' will translate the whole row into a row of a's. If a row starts with 1, then M' will translate the whole row into a row of b's. Now, it is easy to see that $L' := (\{0\}_+ \ominus \{1\} \ominus \{0\}_+) \oplus \{0\}_+^+ \in T(\mathcal{L}(\text{GRFA}))$, while $M'(L') = L$ is not. \square

It would therefore be interesting to study hierarchies of array languages defined by combining array processing devices that alternate between row-wise or column-wise processing, working on multiple passes over given images. This would also enable us to implement error-correction features, like thinning out blurred lines. This also shows that combining different processing modes in subsequent passes could be useful in practice.

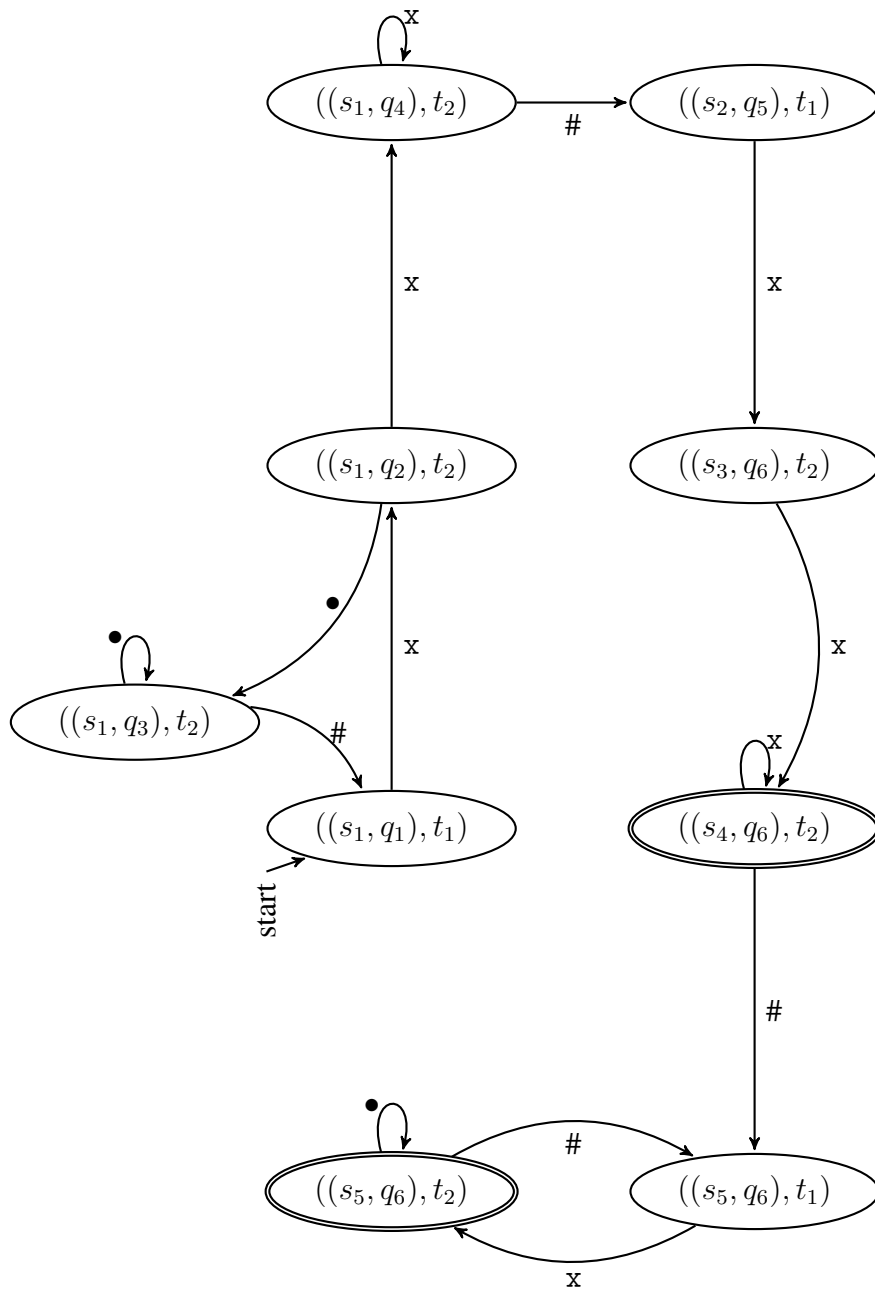


Figure 5.6: GRFA R' that accepts $M^-(L(R^\dagger))$

Chapter 6

Regular Grammars for Array Languages

Several regular-like mechanisms have been proposed in the literature in order to generalize, say, right-linear grammars from the one-dimensional (string) case to the two-dimensional world. RMG and BFA are among the simpler devices. RMG have been extended towards, so-called (regular : regular) array grammars ((R:R)AG) in [106].

As described in Chapter 4, the so-called isometric regular array grammars (IRAG, originally introduced to describe non-rectangular-shaped pictures) can be viewed as natural extensions of BFAs and then seen to describe a superclass of BFA and RMG languages. By splitting the definition of (R:R)AG into two parts, according to the types, we can also formally describe the main reason why (in general) IRAGs cannot be simulated by (R:R)AGs.

6.1 (Regular : Regular) Array Grammars

Siromoney et al. introduced another interesting class of array grammars called (R:R)AG [106] to generate picture languages which cannot be generated by RMG. As we are not so much interested in other language families (as described in [106]), we are now giving a different yet equivalent formalization of this picture language description as we have introduced in [36]:

Definition 57. An (R:R)AG can be specified as $G = (S, V_N, V_I, \Sigma, P_N, P_I, \pi, \tau)$, where the components are as follows:

- A non-terminal alphabet V_N with a distinctive start symbol $S \in V_N$,
- an intermediate alphabet V_I , disjoint from V_N ,

- a terminal alphabet Σ , disjoint from $V_N \cup V_I$,
- a set P_N of non-terminal rules that are either of the form $A \rightarrow XB$ (right-linear), or of the form $A \rightarrow BX$ (left-linear), where $A, B \in V_N$ and $X \in V_I$,
- a set P_I of rules of the form $A \rightarrow X$, with $A \in V_N$ and $X \in V_I$,
- a picture association mapping $\pi : V_I \rightarrow \mathcal{L}_\Sigma(\text{RMG}) \cup \mathcal{L}_\Sigma(\text{BFA})$,
- a type interpretation mapping $\tau : P_N \rightarrow \{\ominus, \oplus\}$ such that $\tau(p_1) = \tau(p_2)$ implies that p_1 is right-linear if and only if p_2 is right-linear.

Observe that the last condition implies that p_1 is left-linear if and only if p_2 is left-linear. The derivation proceeds as follows: first, a derivation tree T is generated by the linear rules given by P_N, P_I , starting with S . According to the type, the inner nodes are henceforth interpreted as row or as column catenation. Finally, π is applied to all leaves of the tree.

So, we obtain a tree whose leaves correspond to array languages and whose inner nodes show catenation operators; hence, we can inductively, bottom-up, associate a language to all inner nodes and hence to the root of T . The language we associate with G is then the union of all languages associated to roots of derivation trees of G in this manner. This describes the language family $\mathcal{L}((R : R)\text{AG})$.

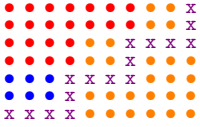
Remark 18. *The definition of $(R:R)\text{AG}$ in [106] is equivalent to the Definition 57. The production rules in [106] have three different forms of rules, say, P_1, P_2 and P_3 , where P_1 is the finite set of non-terminal rules which are regular, P_2 is the finite set of intermediate rules which ensures that the intermediate languages are regular or transpose of such a language, P_3 is the finite set of terminal rules.*

In our Definition 57 we ensure the property of P_1 via τ , the property of P_2 via π , as $\pi(X) = L$ where $L \in \mathcal{L}(\text{RMG})$ or $L \in T(\mathcal{L}(\text{RMG}))$ (i. e., $\mathcal{L}(\text{BFA})$, see Theorem 24) and the property of P_3 also via π .

The conditions in [106] might look more complicated at first glance, due to the fact that there are even three different forms of rules. However, the last two kinds of rules basically allow derivation processes as RMGs do, and our definition is much more similar to the way that $(R:R)\text{AGs}$ are used in [106], [Example 2.1(a)] than the one of that original paper itself.

Example 26. *A $(R:R)\text{AG}$ that generates the staircase of \mathbf{x} 's of a **fixed proportion** is defined as $G = (S, V_N, V_I, \Sigma, P_N, P_I, \pi, \tau)$, where*

- $V_N = \{S, A\}$,
- $V_I = \{X_{\uparrow}, X_{\rightarrow}, X\}$,
- $\Sigma = \{x, \bullet\}$,
- $P_N = \{S \rightarrow AX_{\rightarrow}, A \rightarrow X_{\uparrow}S\}$,
- $P_I = \{S \rightarrow X\}$,
- a picture association mapping $\pi : V_I \rightarrow \mathcal{L}_{\Sigma}(\text{RMG}) \cup \mathcal{L}_{\Sigma}(\text{BFA})$ is given by $\pi(X_{\uparrow}) = \{(\bullet \bullet \bullet)^n \oplus (\bullet) \mid n \geq 1\}$, $\pi(X_{\rightarrow}) = \{(\begin{smallmatrix} \bullet & \bullet & \bullet \\ \bullet & x & \bullet \\ \bullet & x & \bullet \end{smallmatrix}) \ominus (\bullet \bullet \bullet)_n \mid n \geq 1\}$ and $\pi(X) = \begin{smallmatrix} \bullet & \bullet & \bullet & x \\ \bullet & \bullet & \bullet & x \\ x & x & x & x \end{smallmatrix}$, Here $\pi(X_{\uparrow}) \in \mathcal{L}_{\Sigma}(\text{RMG})$, $\pi(X_{\rightarrow}) \in \mathcal{L}_{\Sigma}(\text{BFA})$ and interestingly $\pi(X) \in \mathcal{L}_{\Sigma}(\text{RMG}) \cap \mathcal{L}_{\Sigma}(\text{BFA})$,
- a type interpretation mapping $\tau : P_N \rightarrow \{\ominus, \oplus\}$ is given by $\tau(S \rightarrow AX_{\rightarrow}) = \oplus$ and $\tau(A \rightarrow X_{\uparrow}S) = \ominus$. Here $\tau(S \rightarrow AX_{\rightarrow}) \neq \tau(A \rightarrow X_{\uparrow}S)$.

To obtain  we have the expression $((X_{\uparrow}((X_{\uparrow}X)X_{\rightarrow}))X_{\rightarrow})$, being

interpreted as $((X_{\uparrow} \ominus ((X_{\uparrow} \ominus X) \oplus X_{\rightarrow})) \oplus X_{\rightarrow})$. Let $F_1 = (X_{\uparrow} \ominus X) \oplus X_{\rightarrow}$. We can then provide the recursive expression $F_{n+1} = (X_{\uparrow} \ominus F_n) \oplus X_{\rightarrow}$, $n \geq 1$.

The corresponding tree for F_2 is shown in Fig. 6.1.

Let us define two subclasses of $\mathcal{L}((R : R)\text{AG})$, namely $\mathcal{L}_{\ominus-\ell, \oplus-r}((R : R)\text{AG})$ and $\mathcal{L}_{\oplus-\ell, \ominus-r}((R : R)\text{AG})$.

Definition 58. An $(R:R)\text{AG}$ G with $|\tau(P_N)| = 2$ is called \ominus -left, \oplus -right

- if $A \rightarrow BX \in P_N$ then $\tau(A \rightarrow BX) = \ominus$,
- if $A \rightarrow XB \in P_N$ then $\tau(A \rightarrow XB) = \oplus$.

\ominus -left, \oplus -right $(R:R)\text{AG}$ describes the language family $\mathcal{L}_{\ominus-\ell, \oplus-r}((R : R)\text{AG})$.

Definition 59. An $(R:R)\text{AG}$ G with $|\tau(P_N)| = 2$ is called \oplus -left, \ominus -right

- if $A \rightarrow BX \in P_N$ then $\tau(A \rightarrow BX) = \oplus$,
- if $A \rightarrow XB \in P_N$ then $\tau(A \rightarrow XB) = \ominus$.

\oplus -left, \ominus -right $(R:R)\text{AG}$ describes the language family $\mathcal{L}_{\oplus-\ell, \ominus-r}((R : R)\text{AG})$.

By the definition of $(R:R)\text{AG}$ (see Definition 57), we can state:

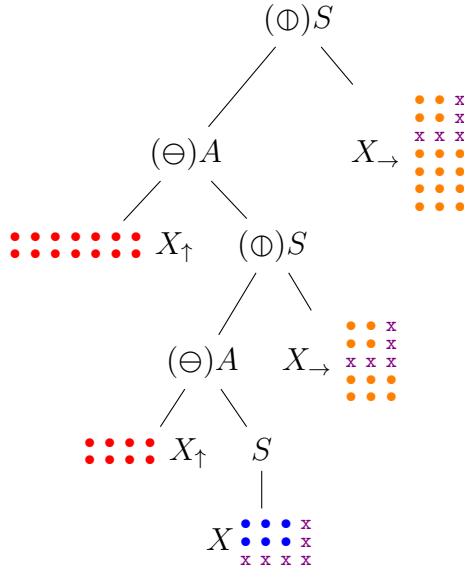




Figure 6.1: Derivation tree for F_2

Lemma 47. $\mathcal{L}((R : R)AG) = \mathcal{L}_{\ominus-l, \oplus-r}((R : R)AG) \cup \mathcal{L}_{\oplus-l, \ominus-r}((R : R)AG)$.

Notice that this is not a partition of $\mathcal{L}((R : R)AG)$, as there are interesting array languages in $\mathcal{L}_{\ominus-l, \oplus-r}((R : R)AG) \cap \mathcal{L}_{\oplus-l, \ominus-r}((R : R)AG)$, as we will see in the following remark.

Remark 19. *For our arguments given in the following, it is crucial to observe that \ominus -left, \oplus -right $(R:R)AG$ differ from \oplus -left, \ominus -right $(R:R)AG$ in the way pictures are puzzled together from basic RMG- or BFA-pieces. Example 26 is generated by some \oplus -left, \ominus -right $(R:R)AG$.*

This can be viewed as starting in the lower left corner of the array, putting  first, then stacking some sub-array on top, some other to the right etc.

But, \ominus -left, \oplus -right $(R:R)AG$ would rather start the puzzling process in the top-most right corner of the array, putting  first, then stacking some sub-array on below that is from $\{(\frac{x}{x}) \oplus (\bullet \bullet \bullet)^n \mid n \geq 1\}$, some other to the left from $\{(\bullet \bullet \bullet)_n \ominus \begin{matrix} \bullet \bullet \bullet \\ \bullet \bullet \bullet \\ \bullet \bullet \bullet \\ x \ x \ x \end{matrix} \mid n \geq 1\}$ and so on.

According to [41], any of the natural way of defining the regularity for two-dimensional objects should coincide along with the classical notion of regularity in some one-dimensional restriction.

Lemma 48. (a) A regular string language corresponds to a language of single-row arrays generated by RMG. (b) A regular string language corresponds to a language of single-column arrays generated by RMG.

Proof. Suppose L is a regular string language. Then, there exists a right-linear grammar $G = (V, \Sigma, P, S)$ such that $L(G) = L$. We can construct an RMG G_1 such that $L(G_1) = L(G) = L$ as follows: $G_1 = (V, V_v, \Sigma_I, \Sigma, S, R^h, R^v)$, where $V_v = \{S_a \mid a \in \Sigma\}$, $\Sigma_I \subseteq V_v$, $R^h = \{A \rightarrow S_a B \mid A \rightarrow aB \in P\} \cup \{A \rightarrow S_a \mid A \rightarrow a \in P\}$ where $A, B \in V$ and $S_a \in \Sigma_I$ and $R^v = \{S_a \rightarrow a \mid S_a \in V_v, a \in \Sigma\}$. Similarly, we can prove (b) by constructing an RMG to generate $T(L)$ in the case of single-column arrays. Here, the intermediate symbols of the RMG will take over the role of the non-terminal symbols of the given right-linear grammar. \square

By combining Lemma 48 with Theorem 24 we could obtain the Corollary 34 and Remark 20, as BFA and RMG languages are contained in $\mathcal{L}_{\ominus-l, \oplus-r}((R : R)AG) \cap \mathcal{L}_{\oplus-l, \ominus-r}((R : R)AG)$.

Corollary 34. (a) A regular string language corresponds to a language of single-row arrays accepted by BFA.

(b) A regular string language corresponds to a language of single-column arrays accepted by BFA.

Remark 20. A language of single-row (column) arrays is in both families $\mathcal{L}_{\ominus-l, \oplus-r}((R : R)AG)$ and $\mathcal{L}_{\oplus-l, \ominus-r}((R : R)AG)$.

Lemma 49. A language of single-row (column) arrays generated by some IRAG corresponds to a regular string language.

Proof. The proof idea is that if we assume the IRAG generating a single-row language not to move down or up then it will be moving either right or left which is nothing but doing the same job of right-linear or left-linear grammar resulting in a regular string language. \square

From previous chapters and also from [30, 35] and [106] we know that $\mathcal{L}(\text{RMG})$ is not closed under row catenation, while $\mathcal{L}(\text{BFA})$ is not closed under column catenation. So, it makes sense to consider the classes $\mathcal{L}(\text{RMG}) \ominus \mathcal{L}(\text{RMG})$ of all array languages that can be represented as row catenations of languages from $\mathcal{L}(\text{RMG})$, as well as $\mathcal{L}(\text{BFA}) \oplus \mathcal{L}(\text{BFA})$.

Lemma 50. $(\mathcal{L}(\text{RMG}) \ominus \mathcal{L}(\text{RMG})) \cup (\mathcal{L}(\text{BFA}) \oplus \mathcal{L}(\text{BFA}))$
 $\subseteq \mathcal{L}_{\ominus-l, \oplus-r}((R : R)AG) \cap \mathcal{L}_{\oplus-l, \ominus-r}((R : R)AG)$.

Proof. Let us consider the rules from some $(\mathbf{R}:\mathbf{R})\mathbf{AG}$, $S \rightarrow X_1A$ and $A \rightarrow X_2$ with either $\pi(X_1), \pi(X_2) \in \mathcal{L}(\mathbf{RMG})$ and $\tau(S \rightarrow X_1A) = \ominus$, or $\pi(X_1), \pi(X_2) \in \mathcal{L}(\mathbf{BFA})$ and $\tau(S \rightarrow X_1A) = \oplus$, we can produce these single catenations with left-linear rules as well and the strictness follows as $(\mathcal{L}(\mathbf{RMG}) \ominus \mathcal{L}(\mathbf{RMG})) \subsetneq \mathcal{L}_{\ominus-\ell, \ominus-r}((\mathbf{R}:\mathbf{R})\mathbf{AG})$ and $(\mathcal{L}(\mathbf{BFA}) \oplus \mathcal{L}(\mathbf{BFA})) \subsetneq \mathcal{L}_{\ominus-\ell, \oplus-r}((\mathbf{R}:\mathbf{R})\mathbf{AG}) \quad \square$

Consider the array language $L = \{0\}_+^+ \oplus \{1\}_+ \oplus \{0\}_+^+$ that can be described by the RMG $G = (V_h, V_v, \Sigma_I, \Sigma, S, R^h, R^v)$, where $V_h = \{S, X, Y\}$, $V_v = \{A, B\}$, $\Sigma_I = V_v$, $\Sigma = \{0, 1\}$, $R^h = \{S \rightarrow AX, X \rightarrow AX, X \rightarrow BY, Y \rightarrow AY, Y \rightarrow A\}$ and $R^v = \{A \rightarrow 0A, A \rightarrow 0, B \rightarrow 1B, B \rightarrow 1\}$. But, there is no BFA for this L , as BFAs are unable to track vertical lines in arbitrary (general) positions, which correspond to the column of 1's in L . Similarly, RMGs cannot describe languages consisting of horizontal lines in arbitrary position. By way of contrast, $L_- = L \ominus \{1\}_+^+$ can be still described by some RMG, although it contains some horizontal line. But this line (row of 1s) is not in arbitrary position, but at the very bottom of all pictures.

A similar argument applies to vertical lines that are (only) at a fixed distant from the top or from the bottom. Slightly modifying the language, we will also say, here and in the following, that, once a grammar was fixed, a line can be in arbitrary position in a sufficiently large picture. The reason is that with respect to the grammar, interchange arguments (based on pigeon hole) will work, as the picture is sufficiently big (a notion that naturally depends on the size of the fixed grammar). This minimum size condition is necessary, as otherwise ‘small arrays’ can be always treated as exceptional arrays by the grammar we look at.

Consider now $L' = L_- \ominus L_- \ominus L$. We claim that L' cannot be written as the row concatenation of any two RMG languages. Assume the contrary. Then, discuss some picture P sufficiently big not to be possibly treated by the hypothetical pair of grammars G_1 and G_2 (with $L' = L(G_1) \ominus L(G_2)$) as a special case. Also, we assume that the two vertical lines are sufficiently far apart in P . Hence, there are $P_1 \in L(G_1)$ and $P_2 \in L(G_2)$ such that $P = P_1 \ominus P_2$. As the two vertical lines are far apart, there must be (at least) one picture, say, P_1 , that is still sufficiently big and contains a vertical line, contradicting interchange arguments. This example proves the strictness of the following inclusion (the inclusion itself is trivial).

Lemma 51. $(\mathcal{L}(\mathbf{RMG}) \ominus \mathcal{L}(\mathbf{RMG})) \subsetneq (\mathcal{L}(\mathbf{RMG}) \ominus \mathcal{L}(\mathbf{RMG}) \ominus \mathcal{L}(\mathbf{RMG}))$.

Corollary 35. $(\mathcal{L}(\mathbf{BFA}) \oplus \mathcal{L}(\mathbf{BFA})) \subsetneq (\mathcal{L}(\mathbf{BFA}) \oplus \mathcal{L}(\mathbf{BFA}) \oplus \mathcal{L}(\mathbf{BFA}))$.

This argument can be generalized to k -fold catenations, as say, P_1 , containing two vertical lines exists simply by pigeon hole.

Theorem 49. *For each $k \geq 1$, we have:*

$\mathcal{L}(\text{RMG})_k \subsetneq \mathcal{L}(\text{RMG})_{k+1}$, as well as $\mathcal{L}(\text{BFA})^k \subsetneq \mathcal{L}(\text{BFA})^{k+1}$.

As (fixed) finite catenations of a single type can be expressed by both language families $\mathcal{L}_{\ominus-\ell, \oplus-r}((\text{R} : \text{R})\text{AG})$ and $\mathcal{L}_{\oplus-\ell, \ominus-r}((\text{R} : \text{R})\text{AG})$, we can also conclude:

Corollary 36. *For any $k \geq 1$,*

$\mathcal{L}(\text{RMG})_k \cup \mathcal{L}(\text{BFA})^k \subsetneq \mathcal{L}_{\ominus-\ell, \oplus-r}((\text{R} : \text{R})\text{AG}) \cap \mathcal{L}_{\oplus-\ell, \ominus-r}((\text{R} : \text{R})\text{AG})$.

In [106, Theorem 3.1], it is claimed that $\mathcal{L}((\text{R} : \text{R})\text{AG})$ is closed under union, without giving any proof. We prove that it is not the case in the following.

Theorem 50. $\mathcal{L}((\text{R} : \text{R})\text{AG})$ is not closed under union.

Proof. Let us reconsider the array language $L = \{0\}_+^+ \oplus \{1\}_+ \oplus \{0\}_+^+$ that can be described by some RMG. Let $\hat{L} = (T(L) \ominus L \ominus T(L)) \oplus L \oplus T(L)$. This language can be described by some (R:R)AG that starts puzzling together the pieces from the left lower corner. Namely, consider the \oplus -left, \ominus -right (R:R)AG specified by: $V_N = \{S, A, B, C, A_T\}$, $V_I = \{X_L, X_T\}$ with $\pi(X_L) = L$ and $\pi(X_T) = T(L)$. Here $L \in \mathcal{L}(\text{RMG})$ and $T(L) \in \mathcal{L}(\text{BFA})$ (by Theorem 24), $P_N = \{S \rightarrow CX_T, C \rightarrow BX_L, B \rightarrow X_TA, A \rightarrow X_LA_T\}$ and $P_I = \{A_T \rightarrow X_T\}$. Notice that

$$H(\hat{L}) = T(L) \oplus L \oplus (T(L) \ominus L \ominus T(L)). \quad (6.1)$$

Assume that there is some (R:R)AG G generating $\hat{L} \cup H(\hat{L})$. Consider a picture P from $H(\hat{L})$ that is sufficiently big and that is generated by G starting in the left lower corner. We also assume that subpictures that compose P by the description of $H(\hat{L})$ are sufficiently big not to be treated as special cases by G .

In process of generating P , there will be first phase where left lower subpicture of P consisting of 0s only is generated (possibly, this gives empty picture), before a BFA- or RMG-image is put on top or to the right of the hitherto produced picture, introducing some 1s for the first time. We have to distinguish two cases.

We give a sample element from $H(\hat{L})$ that explains the two cases in pictorial form in Fig. 6.2.

(i) Assume that the first 1s are generated in a subpicture put on top of the initial block of zeros. This could describe some lower part of subpicture $T(L)$, possibly with some zeros coming from L (on its right). The grammar will proceed, until the first 1s of the long vertical line are produced.

Now, a phase must follow in which all but possibly a few of topmost 1s in this vertical line are produced, as otherwise there is no way to communicate to the position of the 1, which would formally lead to some interchange arguments.

Having derived this, it also means that arbitrarily large pictures from $\{0\}_+^* \oplus (T(L) \ominus L \ominus T(L))$ have to be produced by some RMG or some BFA, which is impossible, as such pictures contain both horizontal and vertical lines in arbitrary position.

(ii) Assume that the first 1s are generated in a subpicture put to the right of the initial block of zeros. If this happened in the first few rows (counted from the bottom border), then this part can still count into the initialization phase.

However, generating 1s of this vertical line later is not possible without running into interchange arguments again, as there is no possibility to communicate the exact position of the vertical line when continuing towards the top border, unless this line (up to few positions) marks the borderline of the generation process. In that case, we arrive at a situation similar as in Case (i), so that we again arrive at a contradiction. \square

From the argument of the previous theorem, in particular observing Eq. (6.1), we conclude:

Corollary 37. *Neither $\mathcal{L}_{\ominus-l, \oplus-r}((R : R)AG)$ nor $\mathcal{L}_{\oplus-l, \ominus-r}((R : R)AG)$ is closed under half-turn.*

By way of contrast, we like to mention the following positive result:

Lemma 52. *$\mathcal{L}_{\ominus-l, \oplus-r}((R : R)AG)$ and $\mathcal{L}_{\oplus-l, \ominus-r}((R : R)AG)$ are closed under union.*

6.2 Regular Grammars, Isometric Array Languages

To determine the relation between (R:R)AGs and IRAGs more precisely, recall the definition of IRAG (see Definition 54). As already mentioned in the introduction of this chapter, BFAs can be interpreted as IRAGs when considering the L_{Rect} interpretation. Namely, BFAs can be seen as IRAGs that are bound to first move (several steps) east, then one step south, then (several steps) west, then one step south, and again (several steps) east, etc. (*)

Remark 21. *As the textbook construction for showing closure under union for grammar-based language families works for IRAGs as well, it is clear that we can*



Figure 6.2: Case (i) (on left) and Case (ii) (on right) for a sample element in $H(\hat{L})$

have a finite number of case distinctions in the constructions for IRAGs that we provide in the following.

Definition 60. Semi-holes are the pattern that has some blank symbol surrounded by three terminal symbols and one blank symbol on its four sides, or two terminal symbols and two blank symbols. Holes are defined to be the pattern that has some blank symbols completely surrounded by terminal symbols or semi-holes.

A picture that explains semi-holes is in Fig. 6.3. Our first non-trivial result states that IRAGs simulating BFAs can be indeed *self-delimiting* in the sense that for the simulating grammar G , $L_{Rect}(G)$ equals the set of terminal arrays derivable in G that contain no holes or semi-holes. Hence, at least the outer shape of such arrays is always rectangular. This is interesting as the rectangle-filter within the definition of $L_{Rect}(G)$ might appear a bit artificial.

Lemma 53. Each BFA can be simulated by some self-delimiting IRAG.

Proof. We can only sketch the basic ideas of the simulation. Assume we are given an IRAG G with the movement restrictions explained in (*) simulating some BFA. Our simulation will work for arrays from a certain number of columns onward only, depending on the number of non-terminals of G . However, IRAGs that are bound to generate arrays with a limited numbers of columns only can easily keep track of this number (within their non-terminals), so that a self-delimiting version of them is straightforward to obtain. Also, if the arrays have only few rows, self-delimiting IRAGs can be built that simulate the generation process, basically by moving first south, then turning around (one step to east), moving north, again one step to east, moving south, etc. The new non-terminals have to keep track of

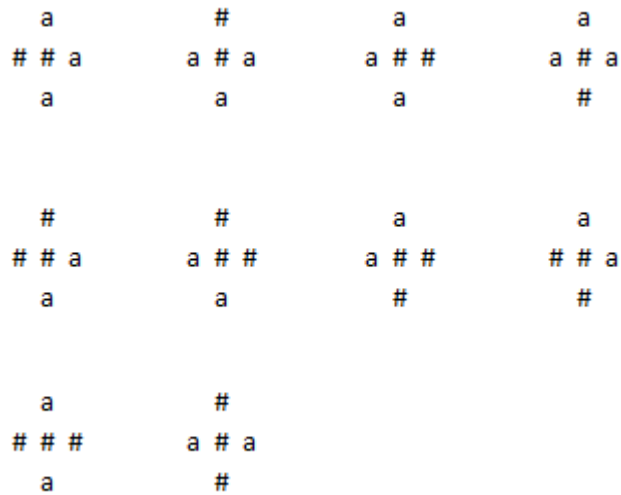


Figure 6.3: Semi-holes

all non-terminal information of the rows that were ‘half-started’, but as there are only few rows to keep track of, this is possible.

We need this minimum number of columns as well as rows, as we are going to codify state information in a certain pattern of shapes. We are explaining this only to some more extent for the right border (that should be straight in the end), but by marking the upper right corner with special coding, one can think of ‘walking around’ the current (already straight) north border in order to stratifying the left border in a similar way as we explain for the eastern (right) border.

Let us (only) describe the grammar that is responsible for generating arrays with a number of rows of the form $6k + 1$ and a number of columns bigger than some number ν like q^6 , where q is the number of states. Assume we are simulating the first move to the right of the BFA (as indicated in the preceding paragraph, this will be rather the second move to the right in the overall simulation). At some point, the generation process guesses to move south, not really completing the first row, actually leaving out two squares to fill. As finally we are going to fill in the missing parts of the array, we have to store within the picture what the non-terminal was when we decided to move down, and what the non-terminal was that we guessed to continue with after moving down. Next, we are moving left by a certain number of steps s that codify both the two states mentioned before and

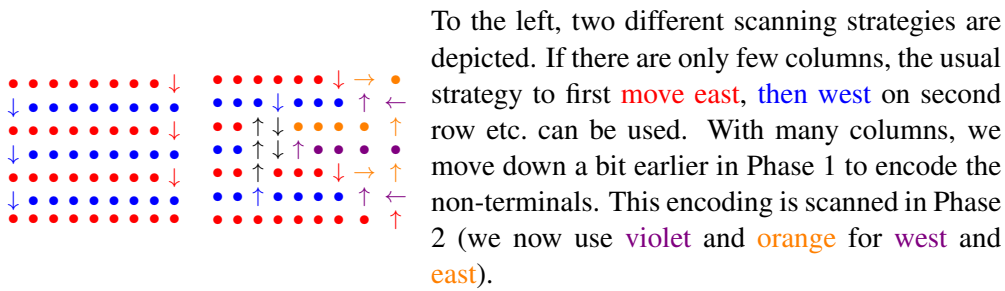
four more states for the next four rows. We are storing the number s that was guessed in our non-terminal, as this allows us to recover the state information of the first six rows. Then, we move down again by three steps to row number five. We move to the east again by s steps, move down on row six and then move back by $s + 1$ steps again, moving upwards until row number two. As s was stored, we can correctly simulate first the transitions on row number six, and then the transitions on the other four rows. Of course, we need to store (again) the correct ‘current’ non-terminal informations for rows two through six.

Having arrived at second row, we continue simulating the BFA as in traditional way, as we know the current non-terminal on the second row, so that we can move left. Having arrived at the left border (whose existence we assume for now), we move south and continue to move eastward, finally verifying that we reached the non-terminal that was guessed for the third row, continuing to move west on the fourth row, east on the fifth row and west again on sixth row, each time verifying the correctness of the guessed non-terminals (as well as the movements made so far) when turning downwards on the east side.

Of course, it could have happened that we turned south prematurely, but this will finally cause some left-over holes or semi-holes. We then continue the whole process in rows seven through twelve, again coding another six-tuple of states. Instead of starting a new cycle when moving eastwards on row $6k + 1$, grammar can likewise decide to terminate. Then, it will start to move up and guess the states for the five penultimate rows that were originally part of the guess when moving down. This guess is now verified by trying to fill in the part that was left over on the move down. Only if the guesses are completely verified, the simulation can continue moving up without leaving holes or semi-holes. Finally, we can end our simulation in the upper rightmost corner of some rectangle around the array or in one of the two neighboring squares. This scanning strategy is also depicted in Figure 6.4. Modifications of this strategy can be developed if the number of rows is not of the form $6k + 1$.

Finally, if the left border is also to be drawn as a straight line, it is clear at least in the small example shown in Figure 6.4 that one could walk in row zero (top row) to the left and, assuming that at the left side a similar encoding strategy was performed as in the right side, then we can again verify this by scanning along the encodings on the left side, as well. \square

For several of the following simulations, a simpler condition is indeed sufficient, assuming that there is some fixed left border. In following statement, the *corner* of a rectangular array is either one of four squares that has two blank neighbors, or



To the left, two different scanning strategies are depicted. If there are only few columns, the usual strategy to first **move east**, then **west** on second row etc. can be used. With many columns, we move down a bit earlier in Phase 1 to encode the non-terminals. This encoding is scanned in Phase 2 (we now use **violet** and **orange** for **west** and **east**).

Figure 6.4: How an IRAG can scan BFA pictures, keeping track of the right border

one of the (in total at most eight) squares of the array that are neighbors of the four previously described squares. This notion of a corner is somewhat more flexible than that of a square having two blank neighbors, and this is necessary because of the well-known chessboard effect: When scanning a rectangular chessboard, one always moves from a black square to a white square or vice versa. Hence, for instance, it is not possible to start in the leftmost upper square on a 8×8 board and move to the rightmost lower square, visiting all squares of the board.

Lemma 54. *Each BFA can be simulated by some IRAG that scans a picture with a fixed left border, starting in any corner and ending in a different corner of the array, assuming that the picture is big enough.*

Proof. Let us consider starting in the leftmost upper square of the array first. We already saw in the previous lemma how to end in the rightmost upper corner. By duplicating the technique to build the right border, it is also possible to move down once more in a third phase, ending up with a straight right side and in the rightmost lower corner. Finally, it would be then also possible to move to the left once more in row $6k + 2$ (assuming the situation described to some detail in the previous proof). By the closure properties of BFA languages, say, regarding reflections, it would be also possible to start scanning in any other corner of the array. Also, only slight modifications are necessary to start the scan in the leftmost upper corner that is not necessary the leftmost upper square according to the preceding discussions. The size restriction in the formulation of the lemma is due to the fact that in particular the scanning of single-row or single-column pictures is not possible under the mentioned conditions. \square

As RMG languages are rotations of BFA languages by [35], analogous statements hold for scanning pictures of RMGs with IRAGs, as well. Also, assuming that one of the borders is already straight, we can maintain the straightness of the other picture borders. We are now ready to present the main result of this section.

Theorem 51. $\mathcal{L}((R : R)AG) \subsetneq \mathcal{L}_{Rect}(IRAG)$.

Proof. Assume in the following (without loss of generality, as we can give such similar arguments otherwise) that we associate row catenation to right-linear rules and consequently column catenation to left-linear rules, a situation depicted in an example in Fig. 6.1. By Remark 19, the picture is puzzled together from pieces of BFA and RMG arrays starting from the leftmost lower of these pictures up to the rightmost upper of these pictures.

One can keep track of derivation, moving bottom-up along the derivation tree, within the non-terminal of the simulating IRAG whose working strategy we are going to sketch now. We are first assuming that all the pictures are big enough for the scanning strategies as described in the previous lemma.

We can start simulation in the leftmost lower corner of the leftmost lower picture and end the scan in the rightmost upper corner of that subarray. Notice that the left border of this subarray is implicitly given by the filter function of the L_{Rect} operator. When doing the scan, we can also guarantee that the rightmost border of this subarray is properly maintained. If the next subarray to be scanned according to the derivation tree is combined via column catenation, then we leave the first subarray either via the rightmost upper square or via the square south to it. Hence, we would enter the second subarray in its leftmost upper corner. As the first array maintained its right border straight, the left border of this second array is straight, as well. We can hence scan the second subarray, guaranteeing all its borders to be straight, and leaving it in the rightmost upper corner. Similarly, if the second subarray would have been combined via row catenation, we would leave the first subarray either via the rightmost upper square or via the square west to it. So, we can scan this second subarray from its rightmost lower to its rightmost upper corner, maintaining all borders straight.

Inductively, we can assume that we are leaving the ‘last’ subarray in its upper right corner, considering the two cases of the row or column catenation as in the previous discussion. As also by induction, the left and lower borders in particular are maintained to be straight, the whole construction will yield a valid scan using the L_{Rect} operator.

Let us briefly discuss the situation when some of the pictures are too small to be scanned in the sketched way. In fact, this is only a formal problem for single-row or single-column pictures. But, it is not hard to see that such exceptional pictures can be merged with neighboring pictures, as (arguing more formally) the language of all single-row or single-column pictures filtered out from some RMG language is also a BFA language and vice versa, so that we can use the known closure properties of these language families to slightly modify the original (R:R)AG to

avoid such situations altogether. The strictness of the inclusion follows from the fact that $\mathcal{L}((R : R)AG)$ is not closed under union whereas $\mathcal{L}_{Rect}(IRAG)$ is closed under union. \square

We would like to mention the following example by attempting to define an IRAG, as it is argued in [106] that no (R:R)AG can generate this example language.

Example 27. Consider the array language

$$L = \left\{ \begin{array}{c} x \\ \bullet \\ x \end{array}, \begin{array}{c} \bullet x \\ x \bullet \\ x x \end{array}, \begin{array}{c} \bullet \bullet x \\ \bullet \bullet x \\ \bullet x x \\ x \bullet x \\ \bullet x x \\ x x x \end{array}, \begin{array}{c} \bullet \bullet \bullet x \\ \bullet \bullet \bullet x \\ \bullet \bullet x x \\ \bullet \bullet x x \\ \bullet x x x \\ x \bullet x x \\ \bullet x x x \\ x x x x \end{array}, \begin{array}{c} \bullet \bullet \bullet \bullet x \\ \bullet \bullet \bullet \bullet x \\ \bullet \bullet \bullet x x \\ \bullet \bullet \bullet x x \\ \bullet \bullet x x x \\ \bullet x x x x \\ \bullet x x x x \\ x \bullet x x x \\ \bullet x x x x \\ x x x x x \end{array}, \begin{array}{c} \bullet \bullet \bullet \bullet \bullet x \\ \bullet \bullet \bullet \bullet \bullet x \\ \bullet \bullet \bullet \bullet x x \\ \bullet \bullet \bullet \bullet x x \\ \bullet \bullet \bullet x x x \\ \bullet \bullet x x x x \\ \bullet x x x x x \\ \bullet x x x x x \\ x \bullet x x x x \\ \bullet x x x x x \\ x x x x x x \end{array}, \dots \right\}$$

being the set of all pictures as given in [106] [Example 2.4]. Let us name (call) the elements in the language L as M_i , $i \geq 1$. So, M_i has i columns. We illustrate the IRAG production rule application process for M_8 and M_9 in Fig. 6.5.

One can see the pattern of the IRAG production rule application process for both odd and even arrays for the general case starting from M_8 and M_9 with the help of the pattern that we have described in Fig. 6.5.

The pattern we have in generation of the pictures in both odd arrays and even arrays ensures that the production rules of the required IRAG has no chance in generating arrays that are not in the language L . We did not give the formal grammar rules as it will have many non-terminals and it is hard to illustrate and write them all in a compact way, we skip the details and instead we give the idea as follows:

First of all, we did not start our pattern from the smallest arrays, instead we did start from M_8 for the even picture and from M_9 for the odd picture, the reason behind is that we need a certain minimum size to make the construction work, but it is easy to give special grammars for the smaller examples that can be finally combined into a bigger grammar for L ; also see Remark 21.

The production rules of the grammar for the larger pictures has 5 phases which are described in following. Notice that there are deterministic, nondeterministic phases. The deterministic phases are basically creating special structures, like keys (colored in blue) and locks (colored in orange and green), that guarantee that no other malicious derivations are possible.

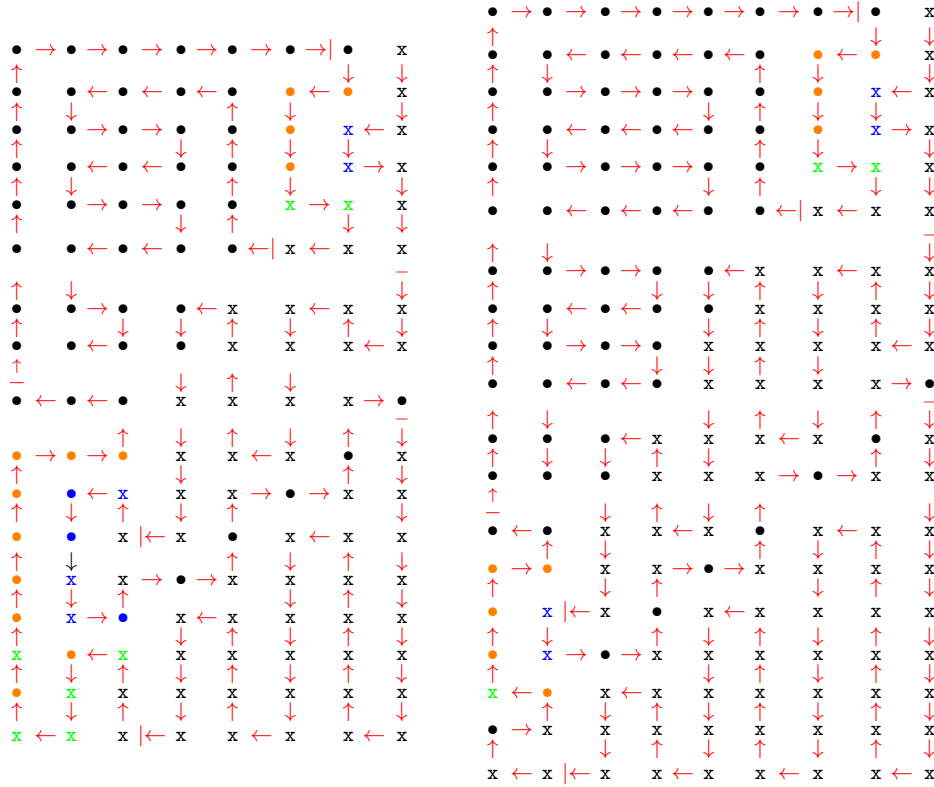


Figure 6.5: How to scan the picture M_8 (on the left) and M_9 (on the right)

The nondeterministic phases fill in the remaining parts of the rectangular shape by sweeping to and fro arbitrarily often. Without such nondeterministic phases, only a finite number of arrays could be created.

*First Phase: This is a deterministic phase, starts from the upper rightmost corner moves two steps down, then makes a **curvy move** in order to create the **first key**, then it goes two steps down again. The end of this phase is shown by — in the pictures in Fig. 6.5.*

Before describing the second phase observe that the picture M_i in this language has a diagonal filled with i \bullet 's separating two regions of x of the give picture, for instance in M_9 it is the diagonal that has nine \bullet 's. Let us call this diagonal a separating diagonal.

Second Phase: This is a nondeterministic phase. It continues after the first phase to create the x's and •'s that are available above the separating diagonal except those that follow the staircase kind of pattern which we see in the next phase. These x's and •'s are created by moving up and down, going west in each change of direction. Also, we can identify two sub-phases: first, only x's are generated, then the sweep southwards is always started by two •'s.

There are two types of guesses in this phase: when moving up or down, it has to be guessed when to change directions, and finally we have to start the next phase. These guesses should be correct, as otherwise there will be holes or semi-holes in the picture, assuming that we are not violating the rectangular outer shape after all. We do not have such arrays in our language. There are minor differences between the generation of •'s in the even and odd case as evident from Fig. 6.5.

Third Phase: In this phase grammar first creates the second key then generates a staircase pattern which is again deterministic up to the decision where to end the staircase. There are some differences between the creation of the keys and in the staircase movements in the even and odd case as evident from Fig. 6.5, where we again marked the beginning and the end of this phase.

Fourth Phase: After finishing the staircase pattern we allow the rules to fill the x's that are available below the separating diagonal except those that were part of the staircase pattern. This is done in a nondeterministic, sweeping fashion, guessing both the times when to turn around and the time when to leave this phase. Wrong guesses will again destroy, desired rectangular structure or leave (semi-) holes.

Fifth Phase: This phase has 4 stages,

in the first stage deterministically we create the second lock as described in the pattern and we mark by strokes where the deterministic phase ends, going beyond the lock structure. We also highlighted the main part of the lock by using orange and green. It should be clear that only the key pattern that was created at the beginning of the third phase fits here. After this deterministic stage we continue with

the second stage, when nondeterministically the grammar rules will generate all the •'s to reach the top left corner and then turn right. This also draws the left border and the upper border of the picture. If the turn is not correctly guessed, the first key-lock structure will be missed which is tested

in the third stage where the rules generate the first lock deterministically (marked with strokes again), and finally

in the 4th stage the grammar rules sweep west-east creating the \bullet 's after the third stage as described in the pattern and stops exactly with required structure next to the second key-lock structure that was described before.

These five phases apply for both odd and even pictures. Now let us justify that the pattern is ensuring the required structure of the pictures in L . We already mentioned the role of deterministic steps that created the two key-lock structures. Moreover, notice that the sweeps of the second and of the fifth phase are both vertically and horizontally designed in a way that prevents deviating from the intended pattern.

We can now also complete our study on the relation of string languages to the array languages studied in this chapter in the following.

Theorem 52. *For a language of single-row (column) arrays L , the following statements are equivalent.*

1. L corresponds to a regular string language.
2. $L \in \mathcal{L}(\text{RMG}) \cap \mathcal{L}(\text{BFA})$.
3. $L \in \mathcal{L}_{\ominus-\ell, \oplus-r}((R : R)AG) \cap \mathcal{L}_{\oplus-\ell, \ominus-r}((R : R)AG)$.
4. L is generated by some $(R:R)AG$.
5. L is generated by some $IRAG$.

Proof. The proof follows like a circular fashion via Lemma 48, Corollary 34, Remark 20, Theorem 51 and Lemma 49. \square

We conclude this chapter by presenting a survey of hierarchical relations between the considered language families in Fig. 6.6 (here arrows mean strict inclusion, arrows that follow by transitivity are omitted, no arrows mean incomparability, otherwise). All these results can be found in this chapter. Notice that $\mathcal{L}(\text{BFA})$ and $\mathcal{L}(\text{RMG}) \ominus \mathcal{L}(\text{RMG})$ are incomparable, as in particular the array language $\{0\}_+^+ \ominus \{1\}^+ \ominus \{0\}_+^+ \ominus \{1\}^+ \ominus \{0\}_+^+ \in \mathcal{L}(\text{BFA}) \setminus (\mathcal{L}(\text{RMG}) \ominus \mathcal{L}(\text{RMG}))$. Similarly, $\mathcal{L}(\text{RMG})$ and $\mathcal{L}(\text{BFA}) \oplus \mathcal{L}(\text{BFA})$ are incomparable.

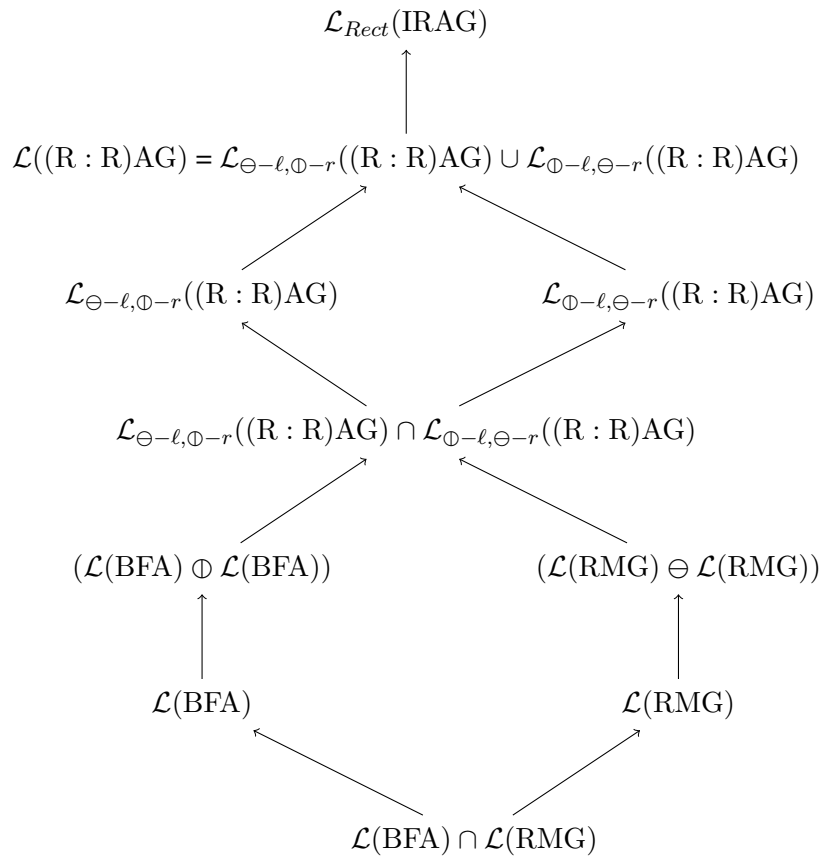


Figure 6.6: The world of rectangular array language families

Chapter 7

Destination: Further Directions

Chapter 2: Eulerian Trails in the Chomsky Hierarchy

We have considered undirected graphs, one can consider directed graphs and find new results in the formal language perspective that we have approached in this chapter. We did not study the colouring of Eulerian trails (graphs) which can be studied further. We have introduced few normal forms in [28] one can continue this research with some or other normal forms, for instance; edge-crossing free Eulerian trails (tours).

We have considered only Eulerian graphs, but one can think of other variants of graphs, for instance; Hamiltonian graphs, Star of David, etc. Also we have not considered 3-dimensional (or n-dimensional) Eulerian graphs (trails) are open for research. Also the concept of trajectories with respect to word representation of graphs can be applied and verified for the (similar) graphs that we have been considered.

We do not have any proper practical applications, implementations, or the real world usage with these concepts that have been introduced in this chapter, for instance; some of the transport modes are Eulerian trails, sometimes tours.

Chapter 3: Jumping Finite Automata

As JFA languages have many more connections with the classical approaches of formal languages [32] considerably deviates and adds more on to [29]. It is hard to trace back all origins and names of the concepts introduced so far. We only mention a few of these sources, also to facilitate finding the names of the concepts and understanding the connections to other parts of mathematics and computer science.

We do not give a survey on all the neighboring areas. Rather, we give some impression about the richness of interrelations in section ‘Discussion of Related Concepts’ in [32]. We have related the concept of jumping finite automata to the, actually quite well-studied, area of expressions involving shuffle operators. This immediately opens up further questions, and it also shows some limitations for this type of research programme.

- Is there a characterization of the class of languages accepted by general jumping finite automata in terms of expressions?¹
- The original motivation for introducing variants of expressions involving shuffle operators was to model the parallel features from the programming languages; see, e. g., [13, 83, 104].

It is well-known that adding all the according features immediately lead to expressions that are computationally complete, i. e., they characterize the recursively enumerable languages [5].

Notably, expressions with limited nesting of iterated catenation, iterated shuffle operators (as provided by our main normal form results for the α -SHUF expressions) have a descriptive power limited by Petri nets (without inhibitor arcs), so that in particular the non-emptiness problem for such limited expressions is decidable (in contrast to the general situation), confer [4, 23, 69, 82].

Yet, decidability questions for Petri nets are quite hard, so that in any case the study of restricted versions of shuffle expressions or related devices is of considerable practical interest. Hierarchies as the one explained in [40] should inspire similar research for α -SHUF expressions.

- The whole area seems to be related to *membrane systems*, also known as *P systems*. The reason is that membrane computing often reduces to *multiset computing*, which is just another name for dealing with subsets of $\mathbb{N}^{|\Sigma|}$. These connections are explained by Kudlek and Mitrana in [73].

Summarizing, the study of expressions involving the shuffle operation, as well as of variants of jumping automata, still offers a lot of interesting questions, as it is also indicated in the recent survey of Restivo [96].

¹We claimed to have found such a characterization at the German Formal Language community meeting in 2014, but this claim turned out to be flawed.

Chapter 4: Scanning Automata and Grammars

Scanning pictures line by line ‘as the ox turns’ is for sure not a new invention in image processing. For instance, in [42], boustrophedon scanning is used as a preprocessing step for the optical character recognition task related to the number plates. Also, this strategy is one kind of the standard way to compose the larger pictures from smaller ones, called *snake by rows*; see https://imagej.net/Image_Stitching.

We have tried to derive a formal model that does mirror this strategy. On the one hand, we have shown that this formal model is pretty stable, as it has various characterizations, and it is even linked to RML, one of the earliest formal models of picture processing. On the other hand, there are quite some natural operations under which we would hope such a model to be closed, as, for example, transpose (which, for BFA, is not the case, see Lemma 42).

There are more powerful models than ours that have been proposed for picture processing, like 4-way NFAs or OTAs; see [41]. However, OTAs are related to our model in the sense that they process a picture diagonal by diagonal, whereas our model process it row by row. The additional power seems to come from the fact that during a computation, OTAs label positions of the pictures by states and this labelling depends not only on the current symbol, but also on the state labels of the upper and left neighbours (i. e., OTAs are special versions of cellular automata). This means that information can be passed from top to bottom in every single column, whereas BFA can only accumulate information of a whole row. Notice that, when we remove this option from the way OTAs work, we arrive at a model that is possibly even closer to ours, the only difference being the way images are scanned. Clearly, diagonal scans can (now) detect diagonal lines, but now there is no way to detect vertical or horizontal ones, as would be the case for RML or BFA.

Conversely, we have seen that diagonals cannot be detected by neither RML nor BFA. Possibly, a more thorough study of different scanning schemes from the point of view of the (typical) classes of images that can be accepted would lead to new insights telling how images should be scanned by computers in practice.

Also in the context of these automata, boustrophedon scanning modes have been discussed, but mostly to study the notion of determinism in connection with Wang tiles; see [76, 77, 78]. The more restricted models of 3-way NFAs or 3-way DFAs do possess a decidable emptiness problem, but the decision procedure seems to be double-exponential, as explained in [93].

As explained in Chapter 4, it is possible to define a restriction of 3-way NFAs (or also 3-way DFAs) that captures (exactly) the class of BFA-pictures.

From a formal language point of view, using two heads, scanning row by row from left to right and right to left in parallel corresponds to even-linear languages as introduced in [2] and further generalized in [38, 25, 108, 112]. We mention bio-inspired models of computation that are closely linked to these language classes, as in [87]. Instead of heads, one could likewise use pebbles for marking.

Also, learnability issues might be of interest, as only very few results are known about learning picture languages, see [109] as one example. In this context, the proven non-existence of (efficiently computable) minimum-state automata poses some problems.

Another idea originating from Inoue, Takanami and Taniguchi [56] is to scan pictures twice, say, first with a BDFA and then let again another BDFA process the quarter-turned picture. Such a processing model is obviously able to accept $\mathcal{L}(\text{RMG}) \cup \mathcal{L}(\text{BFA})$ and enjoys the same decidability properties as BDFAs. It might be good to work this out more in detail, as it could be a quite powerful, yet practically relevant formal model of picture processing. Observe that we get closer to 4-way automata in this way.

As we also touched the area of isometric array languages, it might be good to address the decidability and complexity questions of geometric or topological problems in this context (that cannot be meaningfully done for the non-isometric case), for instance, regarding the number of holes, as this was done in the very beginning of digital picture processing, see [103], but it seems to, ever since, have been neglected, although related combinatorial questions are treated sometimes, see [11].

In both of isometric world and non-isometric world, it might be an idea to approach different discrete topologies of the plane, as frequently discussed in the literature on discrete geometry, for instance, hexagonal or triangular connections; see [70, 88] for recent discussions. Of course, the possible movements of finite automata on such topologies would have to be adapted. As another direction of future research, notice that there have been lots of studies on the descriptonal complexity of regular string languages (see [49]), but not much has been done about a genuinely two-dimensional theory of descriptonal complexity; see [94]. For instance, what about the simple question whether or not the cubic or quadratic blow-ups in the transformations between the models treated (BFAs, RFAs, RMGs, IRAGs) are really necessary?

Lots of research remains to be done here in order to fully understand these simple models of image processing. Finally, our considerations might bear some consequences on several models designed for string processing. We now sketch some connections to so-called sweeping and rotating automata, as discussed in [62].

Sweeping automata process strings by alternating reading directions at the end markers, while rotating automata continue at the left end of the word when it has detected the right end marker. In contrast to two-way automata, they cannot change directions or process positions more often in any other way.

Finally coming back to array languages, let us mention that we are not aware of any other studies on automata minimization apart from the ones that we have mentioned. This opens up venues for future research.

Chapter 5: Picture Transforming Automata

We hope that simple finite automata models that we presented in this chapter can form a starting point to bring these syntactic ideas back into the practice of image processing. This is also why we studied seemingly simplistic working modes of such automata, including their use in image transformations.

We also like to mention that a student of ours has done the implementing of the machine transformation algorithms that we showed to prove closure properties of GBFA languages. This software will be available on request.

A possible further direction of research could be to integrate these models into pattern recognition algorithms. As exhibited by Flasiński in [39], this would be necessitating the development of the Grammatical Inference algorithms. In this context, it looks that we have to overcome the following technical problem: mostly, learners converge to canonical hypotheses like minimum-state automata and hence, efficient learners also comprise efficient state minimization algorithms. However, we have the following theorem in [35]

Theorem 53. *It is NP-hard to decide, given some BDFA $M = (Q, \Sigma, R, s, F, \#, \square)$ on some binary input alphabet $\Sigma = \{a, b\}$, whether or not there is an equivalent BDFA M' with only one state.*

This result seems to pose some difficulties in applying Grammatical Inference techniques to Syntactic Pattern Recognition in the context of picture scanning

automata. It also indicates some limitations to the modular design approach of picture processing automata.

Chapter 6: Regular Grammars for Array Languages

Extending syntactic descriptions from the one- to the two-dimensional world is still an exciting topic, offering quite a number of open questions. In our opinion, it is still unclear what the 'right' extension is. This type of discussion was somehow set to an end by developing quite a number of mutually equivalent mechanisms of regular picture languages in the 1990s.

However, as even the simplest decision problems as mostly as, turn out to be undecidable for these (basically due to the possibility to simulate Turing machine carpets), it is still an open research area to present mechanisms that are simple enough to have nice algorithmic properties and still powerful enough to be useful for practical purposes. Also, understanding better how the various mechanisms interrelate, as undertaken is an approach to classify these mechanisms.

Bibliography

- [1] F. Álvaro, J.-A. Sánchez, and J.-M. Benedí. Recognition of on-line handwritten mathematical expressions using 2D stochastic context-free grammars and hidden Markov models. *Pattern Recognition Letters*, 35:58–67, 2014.
- [2] V. Amar and G. Putzolu. On a family of linear grammars. *Information and Control (now Information and Computation)*, 7:283–291, 1964.
- [3] M. Anselmo, D. Giammarresi, and M. Madonia. New operations and regular expressions for two-dimensional languages over one-letter alphabet. *Theoretical Computer Science*, 340(1):408–431, 2005.
- [4] T. Araki, T. Kagimasa, and N. Tokura. Relations of flow languages to Petri net languages. *Theoretical Computer Science*, 15:51–75, 1981.
- [5] T. Araki and N. Tokura. Flow languages equal recursively enumerable languages. *Acta Informatica*, 15:209–217, 1981.
- [6] M. A. Armstrong. *Groups and Symmetry*. Springer-Verlag, 1988.
- [7] J. Beauquier, M. Blattner, and M. Latteux. On commutative context-free languages. *Journal of Computer and System Sciences*, 35(3):311–320, 1987.
- [8] B. Bérard. Literal shuffle. *Theoretical Computer Science*, 51:281–299, 1987.
- [9] J. Berstel, L. Boasson, O. Carton, J.-E. Pin, and A. Restivo. The expressive power of the shuffle product. *Information and Computation*, 208(11):1258–1272, 2010.
- [10] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. North Holland, 1976.

- [11] V. E. Brimkov, A. Maimone, G. Nordo, R. P. Barneva, and R. Klette. The number of gaps in binary pictures. In G. Bebis, R. D. Boyle, D. Koracin, and B. Parvin, editors, *Advances in Visual Computing, First International Symposium, ISVC*, volume 3804 of *LNCS*, pages 35–42. Springer, 2005.
- [12] M. Cadilhac, A. Finkel, and P. McKenzie. Bounded Parikh automata. *International Journal of Foundations of Computer Science*, 23(8):1691–1710, 2012.
- [13] R. H. Campbell and A. N. Habermann. The specification of process synchronization by path expressions. In E. Gelenbe and C. Kaiser, editors, *Operating Systems OS*, volume 16 of *LNCS*, pages 89–102. Springer, 1974.
- [14] C. Choffrut, A. Malcher, C. Mereghetti, and B. Palano. First-order logics: some characterizations and closure properties. *Acta Informatica*, 49(4):225–248, 2012.
- [15] R. S. Cohen. Star height of certain families of regular events. *Journal of Computer and System Sciences*, 4:281–297, 1970.
- [16] R. S. Cohen and J. A. Brzozowski. General properties of star height of regular events. *Journal of Computer and System Sciences*, 4:260–280, 1970.
- [17] C. R. Cook and P. S.-P. Wang. A Chomsky hierarchy of isotonic array grammars and languages. *Computer Graphics and Image Processing*, 8:144–152, 1978.
- [18] L. C. Eggen. Transition graphs and the star-height of regular events. *The Michigan Mathematical Journal*, 10(4):385–397, December 1963.
- [19] A. Ehrenfeucht, D. Haussler, and G. Rozenberg. On regularity of context-free languages. *Theoretical Computer Science*, 27:311–332, 1983.
- [20] S. Eilenberg and M. P. Schützenberger. Rational sets in commutative monoids. *Journal of Algebra*, 13:173–191, 1969.
- [21] Z. Ésik and W. Kuich. *Modern Automata Theory*. 2012.
- [22] J. Esparza, P. Ganty, S. Kiefer, and M. Luttenberger. Parikh’s theorem: A simple and direct automaton construction. *Information Processing Letters*, 111(12):614–619, 2011.
- [23] J. Esparza and M. Nielsen. Decidability issues for Petri nets – a survey. *EATCS Bulletin*, 52:244–262, 1994.

- [24] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1736. Reprint in [67].
- [25] H. Fernau. Even linear simple matrix languages: formal language properties and grammatical inference. *Theoretical Computer Science*, 289:425–489, 2002.
- [26] H. Fernau and R. Freund. Bounded parallelism in array grammars used for character recognition. In P. Perner, P. Wang, and A. Rosenfeld, editors, *Advances in Structural and Syntactical Pattern Recognition (Proceedings of the SSPR'96)*, volume 1121 of *LNCS*, pages 40–49. Springer, 1996.
- [27] H. Fernau, R. Freund, and M. Holzer. Regulated array grammars of finite index. In Gh. Păun and A. Salomaa, editors, *Grammatical Models of Multi-Agent Systems*, pages 157–181 (Part I) and 284–296 (Part II). London: Gordon and Breach, 1999.
- [28] H. Fernau and M. Paramasivan. Formal language questions for Eulerian trails. In T. Neary and M. Cook, editors, *Machines, Computations and Universality, MCU*, volume 128 of *Electronic Proceedings in Theoretical Computer Science EPTCS*, pages 25–26. Open Publishing Association, 2013.
- [29] H. Fernau, M. Paramasivan, and M. L. Schmid. Jumping finite automata: Characterizations and complexity. In F. Drewes, editor, *Implementation and Application of Automata - 20th International Conference, CIAA*, volume 9223 of *LNCS*, pages 89–101. Springer, 2015.
- [30] H. Fernau, M. Paramasivan, M. L. Schmid, and D. G. Thomas. Scanning pictures the boustrophedon way. In R. P. Barneva, B. B. Bhattacharya, and V. E. Brimkov, editors, *International Workshop on Combinatorial Image Analysis IWCIA*, volume 9448 of *LNCS*, pages 202–216. Springer, 2015.
- [31] H. Fernau, M. Paramasivan, M. L. Schmid, and D. G. Thomas. Simple picture processing based on finite automata and regular grammars. Submitted to *Journal of Computer and System Sciences*, 2017.
- [32] H. Fernau, M. Paramasivan, M. L. Schmid, and V. Vorel. Characterization and complexity results on jumping finite automata. Technical Report arXiv:1512.00482, arXiv, cs.FL, Cornell University, 2015.
- [33] H. Fernau, M. Paramasivan, M. L. Schmid, and V. Vorel. Characterization and complexity results on jumping finite automata. *Theoretical Computer Science*, 679:31–52, 2017.

- [34] H. Fernau, M. Paramasivan, and D. G. Thomas. Regular array grammars and boustrophedon finite automata. In H. Bordihn, R. Freund, B. Nagy, and Gy. Vaszil, editors, *Eighth Workshop on Non-Classical Models of Automata and Applications (NCMA 2016); Short Papers*, pages 55–63, 2016.
- [35] H. Fernau, M. Paramasivan, and D. G. Thomas. Picture scanning automata. In R. P. Barneva, V. E. Brimkov, and J. M. R. S. Tavares, editors, *Computational Modeling of Objects Presented in Images. Fundamentals, Methods, and Applications - 5th International Symposium, CompIMAGE 2016*, volume 10149 of *LNCS*, pages 132–147. Springer, 2017.
- [36] H. Fernau, M. Paramasivan, and D. G. Thomas. Regular grammars for array languages. In R. Freund, F. Mráz, and D. Průša, editors, *Ninth Workshop on Non-Classical Models of Automata and Applications (NCMA)*, pages 119–134, 2017.
- [37] H. Fernau, K. Reinhardt, and L. Staiger. Decidability of code properties. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 41:243–259, 2007.
- [38] H. Fernau and J. M. Sempere. Permutations and control sets for learning non-regular language families. In A. L. Oliveira, editor, *Grammatical Inference: Algorithms and Applications, 5th International Colloquium ICGI 2000*, volume 1891 of *LNCS/LNAI*, pages 75–88. Springer, 2000.
- [39] M. Flasiński. Chapter 1.1; syntactic pattern recognition: paradigm issues and open problems. In C. H. Chen, editor, *Handbook of Pattern Recognition and Computer Vision, 5th Edition*, pages 3–25. World Scientific, 2016.
- [40] N. E. Flick and M. Kudlek. On a hierarchy of languages with catenation and shuffle. In H.-C. Yen and O. H. Ibarra, editors, *Developments in Language Theory, DLT*, volume 7410 of *LNCS*, pages 452–458. Springer, 2012.
- [41] D. Giammarresi and A. Restivo. Two-dimensional languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume III*, pages 215–267. Berlin: Springer, 1997.
- [42] I. Giannoukos, C.-N. Anagnostopoulos, V. Loumos, and E. Kayafas. Operator context scanning to support high segmentation rates for real time license plate recognition. *Pattern Recognition*, 43(11):3866–3878, 2010.
- [43] S. Ginsburg and S. A. Greibach. Principal AFL. pages 308–338, 1970.

- [44] S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.
- [45] S. Ginsburg and E. H. Spanier. AFL with the semilinear property. *Journal of Computer and System Sciences*, 5:365–396, 1971.
- [46] A. C. Gómez and G. I. Álvarez. Learning commutative regular languages. In A. Clark, F. Coste, and L. Miclet, editors, *Grammatical Inference: Algorithms and Applications, 9th International Colloquium, ICGI*, volume 5278 of *LNCS*, pages 71–83. Springer, 2008.
- [47] S. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7:311–324, 1978.
- [48] K. Hashiguchi. Regular languages of star height one. *Information and Control (now Information and Computation)*, 53(3):199–210, 1982.
- [49] M. Holzer and M. Kutrib. Descriptive and computational complexity of finite automata - a survey. *Information and Computation*, 209(3):456–470, 2011.
- [50] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Reading (MA): Addison-Wesley, 1979.
- [51] O. H. Ibarra and S. Seki. Characterizations of bounded semilinear languages by one-way and two-way deterministic machines. *International Journal of Foundations of Computer Science*, 23(6):1291–1306, 2012.
- [52] K. Inoue and I. Takanami. Closure properties of three-way and four-way tape-bounded two-dimensional Turing machines. *Information Sciences*, 18(3):247–265, 1979.
- [53] K. Inoue and I. Takanami. Three-way two-dimensional multicounter automata. *Information Sciences*, 19(1):1–20, 1979.
- [54] K. Inoue and I. Takanami. A note on decision problems for three-way two-dimensional finite automata. *Information Processing Letters*, 10:245–248, 1980.
- [55] K. Inoue and I. Takanami. A survey of two-dimensional automata theory. *Information Sciences*, 55(1-3):99–121, 1991.
- [56] K. Inoue, I. Takanami, and H. Taniguchi. Two-dimensional automata with rotated inputs. *Information Sciences*, 21(3):221–240, 1980.

- [57] K. Inoue, I. Takanami, and R. Vollmar. Three-way two-dimensional finite automata with rotated inputs. *Information Sciences*, 38:271–282, 1986.
- [58] M. Jantzen. Eigenschaften von Petrinetzsprachen. Technical Report IFI-HH-B-64, Fachbereich Informatik, Universität Hamburg, Germany, 1979.
- [59] M. Jantzen. The power of synchronizing operations on strings. *Theoretical Computer Science*, 14:127–154, 1981.
- [60] M. Jantzen. Extending regular expressions with iterated shuffle. *Theoretical Computer Science*, 38:223–247, 1985.
- [61] J. Jędrzejowicz and A. Szepietowski. Shuffle languages are in P. *Theoretical Computer Science*, 250(1–2):31–53, 2001.
- [62] C. A. Kapoutsis, R. Královic, and T. Mömke. Size complexity of rotating and sweeping automata. *Journal of Computer and System Sciences*, 78(2):537–558, 2012.
- [63] J. Kari and V. Salo. A survey on picture-walking automata. In W. Kuich and G. Rahonis, editors, *Algebraic Foundations in Computer Science - Essays Dedicated to Symeon Bozapalidis on the Occasion of His Retirement*, volume 7020 of *LNCS*, pages 183–213. Springer, 2011.
- [64] E. Kinber and C. Smith. *Theory of Computing. A Gentle Introduction*. Prentice Hall, 2001.
- [65] E. B. Kinber. Three-way automata on rectangular tapes over a one-letter alphabet. *Information Sciences*, 35(1):61–77, 1985.
- [66] O. Klíma and L. Polák. On biautomata. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 46:573–592, 2012.
- [67] D. König. *Theorie der endlichen und unendlichen Graphen. Mit einer Abhandlung von L. Euler*, volume 6 of *Teubner-Archiv zur Mathematik*. Leipzig: BSB B.G. Teubner, 1986.
- [68] J. Kortelainen. Remarks about commutative context-free languages. *Journal of Computer and System Sciences*, 56(1):125–129, 1998.
- [69] S. R. Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, STOC*, pages 267–281. ACM, 1982.

- [70] G. Kovács, B. Nagy, and B. Vizvári. On weighted distances on the Khalimsky grid. In N. Normand, J.-P. V. Guédon, and F. Auteursseau, editors, *Discrete Geometry for Computer Imagery - 19th IAPR International Conference, DGCI*, volume 9647 of *LNCS*, pages 372–384. Springer, 2016.
- [71] K. Krithivasan and R. Siromoney. Array automata and operations on array languages. *International Journal of Computer Mathematics*, 4(A):3–40, 1974.
- [72] K. Krithivasan and R. Siromoney. Characterizations of regular and context-free matrices. *International Journal of Computer Mathematics*, 4(A):229–245, 1974.
- [73] M. Kudlek and V. Mitrana. Considerations on a multiset model for membrane computing. In Gh. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing, International Workshop, WMC-CdeA 2002*, volume 2597 of *LNCS*, pages 352–359. Springer, 2003.
- [74] M. Latteux. Cônes rationnels commutatifs. *Journal of Computer and System Sciences*, 18(3):307–333, 1979.
- [75] M. Latteux and G. Rozenberg. Commutative one-counter languages are regular. *Journal of Computer and System Sciences*, 1:54–57, 1984.
- [76] V. Lonati and M. Pradella. Snake-deterministic tiling systems. In R. Kráľovic and D. Niwinski, editors, *Mathematical Foundations of Computer Science, MFCS*, volume 5734 of *LNCS*, pages 549–560. Springer, 2009.
- [77] V. Lonati and M. Pradella. Deterministic recognizability of picture languages with Wang automata. *Discrete Mathematics & Theoretical Computer Science*, 12(4):73–94, 2010.
- [78] V. Lonati and M. Pradella. Strategies to scan pictures with automata based on Wang tiles. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 45(1):163–180, 2011.
- [79] M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and Its Applications*. Reading, MA: Addison-Wesley, 1983.
- [80] A. Mateescu. Scattered deletion and commutativity. *Theoretical Computer Science*, 125(2):361–371, 1994.
- [81] A. Mateescu, G. Rozenberg, and A. Salomaa. Shuffle on trajectories: Syntactic constraints. *Theoretical Computer Science*, 197(1–2):1–56, 1998.

- [82] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984.
- [83] A. W. Mazurkiewicz. Parallel recursive program schemes. In J. Becvár, editor, *Mathematical Foundations of Computer Science 1975, MFCS*, volume 32 of *LNCS*, pages 75–87. Springer, 1975.
- [84] R. McNaughton. The loop complexity of pure-group events. *Information and Control (now Information and Computation)*, 11(1/2):167–176, 1967.
- [85] A. Meduna and P. Zemek. Jumping finite automata. *International Journal of Foundations of Computer Science*, 23(7):1555–1578, 2012.
- [86] A. Meduna and P. Zemek. Chapter 17: Jumping finite automata. In *Regulated Grammars and Automata*, pages 567–585. Springer, New York, 2014.
- [87] B. Nagy. On a hierarchy of $5' \rightarrow 3'$ sensing Watson-Crick finite automata languages. *Journal of Logic and Computation*, 23(4):855–872, 2013.
- [88] B. Nagy. Cellular topology and topological coordinate systems on the hexagonal and on the triangular grids. *Annals of Mathematics and Artificial Intelligence*, 75(1-2):117–134, 2015.
- [89] R. Niedermeier, K. Reinhardt, and P. Sanders. Towards optimal locality in mesh-indexings. *Discrete Applied Mathematics*, 117:211–237, 2002.
- [90] F. Otto. Restarting automata. In Z. Ésik, C. Martín-Vide, and V. Mitrană, editors, *Recent Advances in Formal Languages and Applications*, volume 25 of *Studies in Computational Intelligence*, pages 269–303. Springer, 2006.
- [91] M. Paramasivan and N. G. David. Shuffle operations on Euler graphs. *Malayana Journal of Sciences*, 10(1):63–78, 2011.
- [92] R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.
- [93] H. Petersen. Some results concerning two-dimensional Turing machines and finite automata. In H. Reichel, editor, *Fundamentals of Computation Theory, 10th International Symposium, FCT '95*, volume 965 of *LNCS*, pages 374–382. Springer, 1995.
- [94] D. Průša. Non-recursive trade-offs between two-dimensional automata and grammars. *Theoretical Computer Science*, 610:121–132, 2016.

- [95] P. Prusinkiewicz, A. Lindenmayer, and F. D. Fracchia. Synthesis of space-filling curves on the square grid. In H.-O. Peitgen, J. M. Henriques, and L. F. Penedo, editors, *Fractals in the Fundamental and Applied Sciences*, pages 341–366. IFIP, North-Holland, 1990.
- [96] A. Restivo. The shuffle product: New research directions. In A. Horia Dediu, E. Formenti, C. Martín-Vide, and B. Truthe, editors, *Language and Automata Theory and Applications - 9th International Conference, LATA*, volume 8977 of *LNCS*, pages 70–81. Springer, 2015.
- [97] C. Reutenauer. A new characterization of the regular languages. In S. Even and O. Kariv, editors, *Automata, Languages and Programming, 8th Colloquium, ICALP*, volume 115 of *LNCS*, pages 177–183. Springer, 1981.
- [98] A. Rosenfeld and R. Siromoney. Picture languages – a survey. *Languages of Design*, 1:229–245, 1993.
- [99] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages (3 volumes)*. Springer, 1997.
- [100] H. Sagan. *Space-Filling Curves*. Springer, 1994.
- [101] A. Salomaa. *Formal Languages*. Academic Press, 1973.
- [102] R. Santhanam and K. Krithivasan. Graph splicing systems. *Discrete Applied Mathematics*, 154(8):1264–1278, 2006.
- [103] S. M. Selkow. One-pass complexity of digital picture properties. *Journal of the ACM*, 19(2), April 1972.
- [104] A. C. Shaw. Software descriptions with flow expressions. *IEEE Transactions on Software Engineering*, 4(3):242–254, 1978.
- [105] G. Siromoney, R. Siromoney, and K. Krithivasan. Abstract families of matrices and picture languages. *Computer Graphics and Image Processing*, 1:284–307, 1972.
- [106] G. Siromoney, R. Siromoney, and K. Krithivasan. Picture languages with array rewriting rules. *Information and Control (now Information and Computation)*, 22(5):447–470, 1973.
- [107] G. Siromoney, R. Siromoney, and K. Krithivasan. Array grammars and kolam. *Computer Graphics and Image Processing*, 3:63–82, 1974.

- [108] R. Siromoney. On equal matrix languages. *Information and Control (now Information and Computation)*, 14:133–151, 1969.
- [109] R. Siromoney, L. Mathew, K. G. Subramanian, and V. R. Dare. Learning of recognizable picture languages. In A. Nakamura, M. Nivat, A. Saoudi, P. Shen-Pei Wang, and K. Inoue, editors, *Parallel Image Analysis, ICPIA*, volume 654 of *LNCS*, pages 247–259. Springer, 1992.
- [110] R. Siromoney and K. G. Subramanian. Space-filling curves and infinite graphs. In H. Ehrig, M. Nagl, and G. Rozenberg, editors, *Graph grammars and their application to computer science*, volume 153 of *LNCS*, pages 380–391, 1983.
- [111] K. G. Subramanian, L. Revathi, and R. Siromoney. Siromoney array grammars and applications. *International Journal of Pattern Recognition and Artificial Intelligence*, 3:333–351, 1989.
- [112] Y. Takada. Learning even equal matrix languages based on control sets. In A. Nakamura, M. Nivat, A. Saoudi, P. Shen-Pei Wang, and K. Inoue, editors, *Parallel Image Analysis, ICPIA*, volume 652 of *LNCS*, pages 274–289. Springer, 1992.
- [113] M. Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters*, 30(5):261–264, 1989.
- [114] P. S.-P. Wang. Some new results on isotonic array grammars. *Information Processing Letters*, 10:129–131, 1980.
- [115] I. H. Witten and B. Wyvill. On the generation and use of space-filling curves. *Software–Practice and Experience*, 13:519–525, 1983.
- [116] K. Yanagisawa and S. Nagata. Fundamental study on design system of kolam pattern. *Forma*, 22:31–46, 2007.

Index

- ET° , 32
- $ET_{\text{loop-free}}$, 41
- G_w , 30, 32
- $L_{\text{Rect}}(G)$, 103
- $\Delta(G)$, 17
- α -SHUF expression, 48
- $\delta(G)$, 17
- $\ell(W)$, 17
- \mathbb{N} , 4
- \mathbb{N}_0 , 4
- $\mathcal{D}_{\text{col-f}}$, 136
- $\mathcal{D}_{\text{row-f}}$, 136
- $\nu(G)$, 14
- $\nu(W)$, 18
- $\omega(G)$, 19
- \overline{D} -IRAG, 104
- \overline{L} -IRAG, 104
- \overline{R} -IRAG, 104
- \overline{U} -IRAG, 104
- $\text{perm}(w)$, 46
- $\varepsilon(G)$, 14
- $\varepsilon(W)$, 18
- \oplus -left, \ominus -right (R:R)AG, 152
- \ominus -left, \oplus -right (R:R)AG, 152
- $d_G(v)$, 16
- k -counter machine, 7
- k -cycle, 19
- k -fold column-concatenation, 10
- k -fold row-concatenation, 10
- k -regular graph, 17
- (R:R)AG, 150
- (regular : regular) array grammars, 150
- SHUF expressions, 48
- 3-DFA, 122
- 3-NFA, 122
- adjacency matrix of G , 15
- adjacent vertices (edges), 14
- alphabet, 5
- anti-quarter-turn, Q^{-1} , 10
- anti-transpose, T' , 10
- array, 9
- BFA, 77
- bipartite graph, 15
- blind counter machines, 8
- boustrophedon finite automata, 77
- column concatenation, 9
- column concatenation plus closure, 10
- column product, 10
- complete bipartite graph, 15
- complete graph, 15
- concatenation of walks, 17
- connected graph, 19
- corner, 160
- covering walk, 17
- cycle, 19
- d-BFA, 81
- deterministic k -counter machine, 8
- deterministic finite automaton, 6
- dihedral group, 11
- direction-aware BFA, 81
- disconnected graph, 19
- edge-induced subgraph, 16
- empty graph, 15

empty word, 5
 Euler Trace, ET , 24
 Eulerian Graph, 21
 Eulerian Tour, 20
 Eulerian Trail, 20

 GBFA, 132
 general boustrophedon finite automaton, 132
 general jumping finite automaton, 47
 general returning finite automata, 136
 GJFA, 47
 graph, G , 13
 GRFA, 136

 half-turn, H , 10
 holes, 158
 hull operator, 4

 identical graphs, 15
 incidence matrix of G , 15
 incident edges (vertices), 14
 induced subgraph, 16
 isometric arrays, 101
 isometric regular array grammar (IRAG), 102
 isomorphic graphs, 15
 iterated shuffle, 46

 jumping finite automaton (JFA), 47

 kolam aasanapalakai, 131
 kolam swing plank, 131

 lazy evaluation, 89, 147
 linear, semilinear, 46
 link, 14
 lock, key, 163
 loop, 14

 matrix, 9
 Mealy Picture Machine (MPM), 142
 monoid, 4

 non-isometric arrays, 101
 nondeterministic finite automaton, 7

 parallel edges, 14
 partially blind counter machines, 8
 path, 18
 picture, 9
 picture language, 9
 $PLD_\phi(G)$, 22
 proper subgraph, 16
 pseudo-linear drawing, 22

 quarter-turn, Q , 10

 reflection along horizontal, R_h , 10
 reflection along vertical, R_v , 10
 regular expressions, 7
 regular graph, 17
 regular language, \mathcal{REG} , 7
 regular matrix grammar (RMG), 95
 regular matrix language (RML), 95
 returning finite automata (RFA), 84
 reversal, mirror image, 6
 row concatenation, 9
 row concatenation plus closure, 10
 row product, 10

 section of walk, 18
 self-delimiting IRAG, 158
 semi-holes, 158
 semigroup, 4
 semiring, 5
 separating diagonal, 164
 shape, 101
 shuffle expressions, 48
 shuffle operation, 46
 simple graph, 14
 standard PLD, 32
 star-height, 70
 string form, 71
 subgraph, 16
 subsequence of walk, 18

trail, 18
transpose, T , 10
trivial graph, 15
two-dimensional right-linear grammar,
 95
two-dimensional word, 9

walk, 17
word w of G , 23
word, string, 5