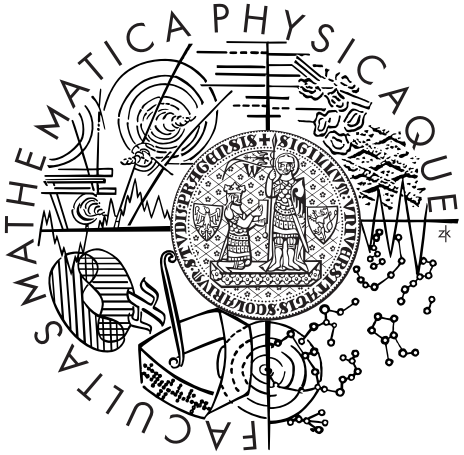
Charles University in Prague
Faculty of Mathematics and Physics

DOCTORAL THESIS



Petr Hoffmann

Machine Learning of Analysis by Reduction

Department of Software and Computer Science Education

Supervisor of the doctoral thesis: František Mráz
Study programme: Computer Science
Specialization: I1 — Theoretical Computer Science

Prague 2013

I would like to thank my supervisor František Mráz and my family for their patience and support.

I would like to thank following entities for supporting my research work. My work was partially supported by:

- the Grant Agency of Charles University in Prague under Grant-Nos. 120709, 358/2006/A-INF/MFF, 300/2002/A-INF/MFF,
- the Grant Agency of the Czech Republic under Grant-Nos. 201/04/2102, 201/02/1456, and
- the program Information Society under project 1ET100300517.

Especially the first mentioned project (Grant-No. 120709, called *Test methods for grammar inference algorithms*) brought many important theoretical and practical results presented in this thesis.

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, 15. 04. 2013 signature of the author

Název práce: Strojové učení redukční analýzy

Autor: Petr Hoffmann

Katedra: Kabinet software a výuky informatiky

Vedoucí disertační práce: RNDr. František Mráz CSc., Kabinet software a výuky informatiky

Abstrakt: Práce se zabývá učením modelů redukční analýzy, která je důležitým nástrojem pro zpracování vět přirozeného jazyka. Dokazujeme, že hledání malých modelů na základě pozitivních a negativních příkladů je NP-těžké oproti úloze uvažující pouze pozitivní příklady, pro kterou navrhujeme efektivní algoritmus. Navrhujeme model redukční analýzy (tzv. single k-reversibilní restartovací automat) a metodu pro jeho učení z pozitivních příkladů redukčních analýz. Ukazujeme, že síla tohoto modelu leží mezi rostoucími kontextovými jazyky a kontextovými jazyky. Dále navrhujeme metodu pro testování učících algoritmů, která pracuje s cílovými jazyky založenými na náhodných automatech. Ta je následně použita na otestování naší učící metody. Navíc ukazujeme několik omezení testovacích metod používajících cílové jazyky založené na gramatikách.

Klíčová slova: redukční analýza, restartovací automaty, gramatická inference, testování

Title: Machine Learning of Analysis by Reduction

Author: Petr Hoffmann

Department: Department of Software and Computer Science Education

Supervisor: RNDr. František Mráz CSc., Department of Software and Computer Science Education

Abstract: We study the inference of models of the analysis by reduction that forms an important tool for parsing natural language sentences. We prove that the inference of such models from positive and negative samples is NP-hard when requiring a small model. On the other hand, if only positive samples are considered, the problem is effectively solvable. We propose a new model of the analysis by reduction (the so-called single k-reversible restarting automaton) and propose a method for inferring it from positive samples of analyses by reduction. The power of the model lies between growing context-sensitive languages and context-sensitive languages. Benchmarks using targets based on grammars have several drawbacks. Therefore we propose a benchmark working with targets based on random automata, that can be used to evaluate inference algorithms. This benchmark is then used to evaluate our inference method.

Keywords: analysis by reduction, restarting automata, grammatical inference, benchmark

Contents

Introduction

In this thesis we will work with formal languages. A formal language is a set of words, which are finite sequences of letters called symbols. However, in this introduction we use the natural language terminology, i.e. by a *word* we mean entities such as "boy", "table", "go", etc. A *sentence* means an arbitrary sequence of words, e.g. "Peter is eating an apple", "Peter is eats an apple", or "apple apple Peter". Thus, we consider both the syntactically correct (according to some widely accepted English grammar) and syntactically incorrect sequences of words to be sentences. Finally, a *language* means *any* set of sentences. Some sample languages are given below:

- L_1: the empty language, i.e. the empty set of sentences,
- L_2: English, that is the set of all correct English sentences,
- L_3: the set of all syntactically incorrect English sentences,
- L_4: the set of all sentences used in this thesis, and
- L_5: the set of all words representing natural numbers (i.e. the set of words like "one", "two", etc.).

It is often helpful to have a device able to decide whether a given sentence belongs to some given language. For example, in the case of L_2 (the English), it should decide whether a given sentence is syntactically correct or not. Even more useful would be a device able to parse any given sentence, to reveal its structure in the case of a correct sentence, or to locate syntactical errors and possibly to recover the correct sentence from its incorrect version.

For some languages (like L_1 and L_4), it would be very easy to find such a device. For others, it is a challenging task nowadays (consider L_2 (the English) — surely there is even a lot of people still unable to decide whether a given complex sentence is correct or not). So building such a device is considered to be a hard task.

On the other hand, we daily see small children able to learn a large portion of their mother tongue from sample sentences and simple hints. There we can see a pair of a *learner* (i.e. the child) and a *teacher* (i.e. the environment — parents, schoolmates, TV shows, etc.). The teacher is a source of sample sentences, the so-called *samples*, and possibly some other information. Sometimes the sample sentence is *labeled* as either *a positive sample*, e.g. it is claimed to be syntactically correct, or as *a negative sample*, e.g. as being syntactically incorrect. In this setting the children are *learning* their mother tongue through building their own *model* of that language. Similar scenarios are present in fields such as artificial intelligence, pattern recognition, etc. (see e.g. [54] for a brief introduction), where the ability to learn from labeled samples is used to classify all sorts of unseen samples later.

The just introduced problem of learning languages based on given input samples is known as the problem of *grammatical inference*. It is said that a model is being *inferred* from given samples. There are several approaches to learning

languages, regarding different forms of samples, different ways of obtaining samples (one can for example get a sample based on a previous question), etc. Some common approaches are presented below in Chapter 2. For now we will proceed with the basic idea given above.

One can try to describe a language using several kinds of models, e.g. just a list of all allowed sentences (for finite languages), grammars, automata, or one's brain. In grammatical inference we also have to choose the class of models to be considered. This choice significantly restricts the class of languages that can be learnt and also the size of the search space (we can see learning as a search for the right language model).

The initial inspiration for this thesis comes from the field of linguistics. There the so-called *analysis by reduction* (ABR for short) [37, 49, 64] forms an important tool for parsing natural language sentences. It is popular especially among European linguists. It can be used to check sentences for correctness, to find dependencies present in them, and to locate errors. The ABR consists in a stepwise simplification of a given extended sentence until a simple sentence is obtained or an error is found, as illustrated below [37] (the underlined words are deleted or rewritten):

Martin, Peter and Jane work very slowly.
⇓
Peter and Jane work very slowly.
⇓
Jane works very slowly.
⇓
Jane works slowly.
⇓
Jane works.

As you can see, the input sentence is iteratively simplified by either just removing its part or by replacing its part with another shorter part. Finally, we obtain a simple sentence. As we ended with a correct sentence (note that a simple sentence can be easily checked to be correct) and individual simplifications are not allowed to remove errors, we conclude that the original input sentence is correct as well.

For many languages, we are able to perform the ABR by hand. The ultimate goal is to perform it automatically. However, this requires a model of the ABR for those languages. Thus, we would like to find models for given languages (or, more precisely, for their ABR) if possible.

In order to model the ABR (and thus to model also the corresponding language), one can use the so-called *restarting automaton* [39, 60, 59]. A restarting automaton is a model describing a set of simplification rules and a set of simple sentences to be used in the ABR for a given language. We can use such a model of the ABR to describe a language — the set of sentences, which can be simplified to the simple sentences given in the model using the simplification rules of the model. So we can use a restarting automaton as a model of a target language in grammatical inference. By inferring a restarting automaton, we infer also the represented language, of course.

Now we can briefly summarize the goal of this thesis — we will study inference of restarting automata from samples describing unknown target languages. More precisely, we will learn from samples of the ABR, but only samples of the ABR of syntactically correct sentences will be considered — it can be seen as a special case of learning from positive samples only. The proposed solution will be then evaluated on a new benchmark, suitable for evaluating methods for inferring the ABR. The design of this benchmark forms an important part of this thesis.

The inference of restarting automata has been studied for several years. There were some attempts to learn restarting automata using genetic algorithms [12, 13, 32]. However, the achieved results are far from being applicable. A method of learning restarting automata from positive samples only was proposed in [55]. However, tests on neither artificial languages nor real ones were presented. Moreover, a very helpful teacher was assumed. Nevertheless, this method significantly influenced our one as will be detailed in Chapter 6.

We already said that we consider learning from positive samples only as well. The task to learn from positive samples only is quite complex. Particularly, Gold [24] showed that any class of languages containing all finite sets and at least one infinite set is not learnable in the limit from positive data only. To overcome the non-learnability, we enrich the input of the grammatical inference problem described above (i.e. sample sentences only) with correct samples of the ABR. This means we show the learner how to process given sentences in order to simplify them to simple sentences. We hope it will significantly help the learning algorithm to identify the correct simplifications to be performed.

For example, in a more common problem setting, the input would consist of sample sentences only — for English, the input could consist of the following two sentences:

a sample sentence: Jane works very slowly.

a sample sentence: Peter lies.

However, in our approach we consider the following input:

a sample of the ABR: Jane works very slowly. $\Rightarrow$ Jane works slowly. $\Rightarrow$ Jane works.

a sample of the ABR: Peter lies.

For those who are used rather to formal languages, we present below an example input for learning the language $\{0^n1^n; n \geq 0\}$ (note that in this particular case, the basic elements are symbols and words in contrast to the previous example working with words and sentences, respectively — this corresponds to the common terminology used in the field of formal languages theory). Below we present the input consisting of just one sample of the ABR — however, note that four words are claimed to belong to the language and three examples of the simplification relation are given. So a single sample of the ABR may contain much more information than a sample word!

a sample of the ABR: $000111 \Rightarrow 0011 \Rightarrow 01 \Rightarrow \lambda$.

In order to identify the right place for performing a simplification, restarting automata usually use constraints in form of regular languages. Various other approaches were studied, too. E.g. in [55, 13] the so-called strictly locally testable languages [73, 42] were used instead of regular languages in those constraints. Even more restricted versions of restarting automata were proposed by Basovník and Mráz [10], where only fixed size contexts around the rewritten subword is considered. The inference of such automata was done by genetic algorithms. Even more restricted model, the so-called clearing restarting automata was proposed by Černo and Mráz [15, 16]. A clearing restarting automaton not only considers fixed size contexts but also restricts rewriting into deleting some continuous part of the given word only. Its learning from both positive and negative samples was presented in [14]. Our approach is similar to [55], we consider another class of languages to be used in constraints, namely the class of reversible languages [4]. In contrast to regular languages, the reversible languages can be inferred from positive samples [4] as in the case of strictly locally testable languages. Moreover, the class of strictly locally testable languages is known to be a proper subclass of the class of reversible languages [42]. The actual usability of our approach needed further research that is present in this thesis as well. Considering those classes of languages enables us to learn from syntactically correct sample sentences only, i.e. no samples containing errors are needed. By restricting constraints used for restarting automata to k-reversible languages, we obtain a new model of restarting automata.

We analyze the properties of the proposed model, particularly we estimate its power and we also prove that there is a hierarchy of such models with respect to the degree of reversibility k. We design an algorithm for inferring those models (i.e. models of the ABR) from positive samples. To evaluate our inference algorithm, we show its performance on certain sample languages used by other researchers. To obtain statistically more relevant results, we needed a method able to supply random samples in form of samples of the ABR. There are several possible ways to design such benchmarks. We prove some theoretical limits of generating random samples of the ABR using context-free grammars, and we finally propose another benchmark which generates random samples of the ABR from randomly generated restarting automata.

According to the experimental results, our method performs really well when inferring both the fixed languages used by other researchers and the languages represented by randomly generated restarting automata.

This thesis is closely related to the following papers written by the author of this thesis. Some results presented here were taken directly from those works. We started the study of learning restarting automata in [34, 35, 32] where the possibility of using genetic algorithms for learning from both positive and negative sample words and positive and negative samples of simplifications was considered. In [35] we also presented a method that can be used to evaluate algorithms for inferring restarting automata. A deeper analysis of such benchmarks was given in [29]. A simple method for learning a restricted version of restarting automata was presented and briefly evaluated in [31]. Finally, the complexity of inferring restarting automata was analyzed in [33].

Below we present a list of our papers mentioned in this thesis:

[29] HOFFMANN, P. Evaluation of analysis by reduction inference algorithms.

In *AIA 2010 – Artificial Intelligence and Applications*. Calgary : Acta Press, 2010. pp. 67–74. ISBN 9780889868175.

[30] HOFFMANN, P. Improving RPNI algorithm using minimal message length. In *Proceedings of the 25th IASTED International Multi-Conference: Artificial Intelligence and Applications, AIAP'07*. Anaheim : Acta Press, 2007. pp. 378–383. ISBN 9780889866294.

[31] HOFFMANN, P. Learning analysis by reduction from positive data using reversible languages. In *Proceedings of the 2008 Seventh International Conference on Machine Learning and Applications, ICMLA '08*. Washington : IEEE Computer Society, 2008. pp. 141–146. ISBN 9780769534954.

[32] HOFFMANN, P. Learning restarting automata by genetic algorithms. In BIELIKOVÁ, M. (ed.). *SOFSEM 2002: Student research forum, Milovy, Czech Republic*. Milovy : Slovak technical university, 2002. pp. 15–20.

[33] HOFFMANN, P. *Restarting automata inference complexity*. Technical Report TR 2007/9. Prague : Faculty of Mathematics and Physics, Charles University, 2007. 10 p.

[34] HOFFMANN, P. *Učenie reštartovacích automatov genetickými algoritmami*. Bachelor's thesis. Prague : Faculty of Mathematics and Physics, Charles University, 2000. [*In Czech, the English translation of the title is Learning restarting automata by genetic algorithms.*]

[35] HOFFMANN, P. *Učenie reštartovacích automatov genetickými algoritmami*. Master's thesis. Prague : Faculty of Mathematics and Physics, Charles University, 2003. 129 p. [*In Czech, the English translation of the title is Learning restarting automata by genetic algorithms.*]

The text of this thesis is structured as it follows. In Chapter 1 we introduce elementary results from the field of formal languages and from the theory of graphs that will be heavily used in the rest of this thesis. Chapter 2 summarizes related work, particularly approaches to learning languages in general and approaches to learning of the ABR. Goals of this thesis are summarized in Chapter 3. Formal definitions of the problems studied in later chapters are given in Chapter 4. Next, in Chapter 5 we study the complexity of some selected problems from the area of the ABR learning. In Chapter 6 we propose a new method for learning the ABR from positive samples. For evaluating the proposed algorithm, we designed a benchmark described in Chapter 7 where the proposed method is evaluated as well. In Conclusion, some problems present in the current approach are discussed and future work is proposed.

1. Preliminaries

This chapter introduces necessary definitions and theorems that will be used in the following text. In Section 1.1 we present basic concepts of the theory of formal languages such as a word, a language, a finite state automaton, etc. Additional concepts mainly from the theory of graphs are given in Section 1.2.

Throughout the thesis, we use $\mathbb{N}, \subseteq, \subset$ and $\subset_{\text{fin}}$ for the set of non-negative integers (i.e. including zero), the set inclusion relation, the proper set inclusion relation and the finite subset relation, respectively. By $\mathrm{P}(X)$ we denote the set of all subsets of the set X. For two sets X and Y, we denote $X \cup Y, X \cap Y$, and $X \setminus Y$ the union, the intersection, and the difference of X and Y, respectively. For two sets X and Y, their symmetric difference is defined as $X \oplus Y = (X \cup Y) \setminus (X \cap Y)$.

A brief list of used symbols can be found on page 142.

1.1 Formal Languages Basics

Here we introduce some elementary concepts of the formal languages theory together with some more advanced topics. For a more complete presentation, see e.g. [36, 48].

Definition 1.1.1. *An* alphabet *is a finite set of elements called* symbols.

We often use symbols like Σ, Γ to denote alphabets. Variables for symbols are usually denoted by a, b, c, d, while for particular symbols we use only digits written as $0, 1, 2$, or special symbols such as $\bullet, \circ, \oplus, \otimes$. A sequence of symbols forms a word:

Definition 1.1.2. *Let Σ be an alphabet. A* word *over Σ is a finite sequence of symbols from Σ. The word containing no symbols is called the* empty word *and it is denoted by λ. The length of the sequence is called the* length of the word *and is denoted by $|w|$ for a word w.*

By $|w|_a$ (where w is a word over Σ and $a \in \Sigma$ is a symbol) we denote the number of occurrences of the symbol a in the word w.

Using symbols 0 and 1, we can form words like 001, i.e. we simply concatenate the symbols as the symbols can be easily identified and therefore there is no risk of an ambiguity. We usually use symbols from the end of the alphabet to denote variables for words, e.g. u, v, w, x, y, z. Finally, a language is an arbitrary set of words:

Definition 1.1.3. *Let Σ be an alphabet. A* language L over Σ *is any set of words over Σ.*

We use upper case letters like L to denote languages.

Note the similarity to the approach presented in Introduction. The terms a word and a sentence used there correspond to the just defined terms a symbol (i.e. a symbol in a formal language represents a word in a natural language), and a word (i.e. a word in a formal language represents a sentence of a natural language), respectively. In the rest of this thesis, we will use the latter terms.

Next, we define some operations on words and languages.

Definition 1.1.4. *Let u and v be two words. By $u \cdot v$ or just by uv we denote the word obtained by concatenating the two sequences of symbols, and we call the result the* concatenation of words u and v.

Let L_1 and L_2 be two languages. A concatenation of L_1 and L_2 *is the language* $L_1 \cdot L_2 = \{w_1 \cdot w_2; w_1 \in L_1, w_2 \in L_2\}$.

Concatenating a language with itself gives us some useful languages:

Definition 1.1.5. *Let L be a language over Σ. By L^0 we denote the language $\{\lambda\}$. Then we define further concatenations recursively as $L^{n+1} = L^n \cdot L$ for $n \geq 0$. Finally, $L^* = \bigcup_{n \in \mathbb{N}} L^n$ is the set of all possible concatenations of words from L (this operation is called Kleene closure). Similarly, we define $L^+ = \bigcup_{n \in \mathbb{N} \setminus \{0\}} L^n$.*

As a special case, we often use the language obtained by concatenating an alphabet Σ with itself several times. As Σ could be taken as a language over Σ, the meaning of Σ^k for $k \geq 0$ is clear from the definition above. For example, we have a set Σ^3 of all the words over Σ having the length exactly 3. Also note that the definition of a language over Σ corresponds exactly to a subset of Σ^*. It is also sometimes useful to consider only words having some minimal required length:

Definition 1.1.6. *Let Σ be an alphabet, $k \geq 0$. By $\Sigma^{\geq k}$ we denote $\bigcup_{n \geq k} \Sigma^n$.*

We may be interested in exactly the words not being members of some language. The so-called complement of a language is defined as it follows:

Definition 1.1.7. *Let L be a language over Σ. By L^{c} we denote its* complement to Σ^*, *e.g. $L^{\mathrm{c}} = \Sigma^* \setminus L$.*

It is sometimes useful to consider words obtained by reverting the order of symbols in another word:

Definition 1.1.8. *Let w be a word over some alphabet Σ. We define the reverse of w (denoted w^{R}) as follows:*

- $\lambda^{\mathrm{R}} = \lambda$, *and*
- $(ua)^{\mathrm{R}} = a(u)^{\mathrm{R}}$ *for all* $u \in \Sigma^*, a \in \Sigma$.

Let $L \subseteq \Sigma^*$ be a language. We define the reverse of L as $L^{\mathrm{R}} = \{w^{\mathrm{R}}; w \in L\}$.

To easily manipulate words in theorems, definitions, etc., we introduce the following notation:

Definition 1.1.9. *Let $w \in \Sigma^*$ be a word and let $\{i_0, \ldots, i_n\} \subset \mathbb{N}$ be a set of numbers (where $0 \leq i_0 < i_1 < \ldots < i_n \leq |w| - 1$) for some $n \in \mathbb{N}$. We denote $w_{\overline{i_0,\ldots,i_n}}$ the word obtained from w by removing symbols having the specified positions $i_0, \ldots, i_n$ (the index of the first symbol being zero).*

It is possible to obtain a word by shifting symbols in some other word — by performing the so-called rotation:

Definition 1.1.10. *Let Σ be an alphabet and let $w \in \Sigma^*$ be a word. A* rotation $w \lll n$ of w by n symbols *(where $n \in \mathbb{N}$) is defined as follows:*

- $w \lll 0 = w$ *for all* $w \in \Sigma^*$,
- $\lambda \lll n = \lambda$ *for all* $n \in \mathbb{N}$,
- $aw \lll n = wa \lll (n-1)$ *for* $a \in \Sigma, w \in \Sigma^*$ *and* $n > 0$.

The set of all rotations of $w \in \Sigma^*$ *is the set* $\{w \lll n; n \in \mathbb{N}\}$ *denoted by* $\mathrm{rot}(w)$.

Note that the set of all rotations of any word w contains only finitely many words.

There are several approaches to representing a language. The basic ones are automata and grammars. We will introduce automata at first. One can see an automaton as a device reading an input word and returning 1 if the word was accepted by the automaton, and returning 0 if the word was rejected by the automaton. Then, exactly the accepted words form the represented language. Different types of automata have different power regarding their ability to represent languages. Here we will consider the so-called finite state automata, which belong to the less powerful models for representing languages. We often use symbols like A, B, M to denote automata.

Definition 1.1.11. *A* finite state automaton *(FSA) is a quintuple* $A = (Q, \Sigma, \delta, I, F)$, *where*

- Q *is a finite set of states,*
- Σ *is an alphabet,*
- $\delta : Q \times \Sigma \to \mathrm{P}(Q)$ *is a transition function,*
- $I \subseteq Q$ *is a set of initial states, and*
- $F \subseteq Q$ *is a set of accepting states.*

A finite state automaton A *is called* deterministic *(DFSA), if for each state* $q \in Q$ *and symbol* $a \in \Sigma$ *it holds* $|\delta(q, a)| \leq 1$ *and* $|I| = 1$. *Otherwise, the automaton is called* non-deterministic *(NFSA). Moreover, a DFSA* A *is called* complete, *if for each state* $q \in Q$ *and symbol* $a \in \Sigma$ *it holds* $|\delta(q, a)| = 1$.

We extend the transition function δ *to* $\delta^* : P(Q) \times \Sigma^* \to P(Q)$ *as follows:*

- $\delta^*(\emptyset, w) = \emptyset$ *for all* $w \in \Sigma^*$,
- $\delta^*(R, \lambda) = R$ *for all* $R \subseteq Q$, *and*
- $\delta^*(R, au) = \delta^*((\bigcup_{r \in R} \delta(r, a)), u)$ *for all* $R \subseteq Q, R \neq \emptyset$ *and* $a \in \Sigma, u \in \Sigma^*$.

A word w *is* accepted *by a finite state automaton* A *if* $\delta^*(I, w) \cap F \neq \emptyset$. *The set of words accepted by* A *is called the* language accepted by A *and is denoted as* $\mathrm{L}(A)$.

There can be a special state in a finite state automaton that can not be left. We define it below:

Definition 1.1.12. *Let $A = (Q, \Sigma, \delta, I, F)$ be a finite state automaton. A state $d_0 \in Q$ is called a dead state if $\delta(d_0, a) = \{d_0\}$ for each $a \in \Sigma$.*

In general, the term *finite state automaton (FSA)* refers to either a DFSA or a NFSA. A well-known result [36] says that DFSA's and NFSA's accept the same class of languages, namely the class of the so-called *regular languages*. Thus, we can use finite state automata for representing regular languages.

Definition 1.1.13. *The class of languages accepted by DFSA's is called the class of* regular languages*.*

It is sometimes useful to work with a set of prefixes of words from some language. We therefore introduce the following notation.

Definition 1.1.14. *Let $L \subseteq \Sigma^*$ be a language. We define the set* $\mathrm{Pr}(L) = \{u; \exists v \in \Sigma^*$ *such that* $uv \in L\}$.

In the following text, we will need the so-called left-quotient of L and a word w defined as follows.

Definition 1.1.15. *Let $L \subseteq \Sigma^*$ be a language and let $u \in \Sigma^*$ be a word. We define the* left-quotient of L and u *by* $\mathrm{LQ}(L, u) = \{v \in \Sigma^*$ *such that* $uv \in L\}$.

This idea can be further extended to the so-called left-quotient of a language with another language.

Definition 1.1.16. *Let $L_1, L_2 \subseteq \Sigma^*$ be two languages. The* left-quotient of L_1 with L_2 *is defined as* $\mathrm{LQ}(L_1, L_2) = \{v; uv \in L_1, u \in L_2\}$.

Note that $\mathrm{LQ}(L, u) = \mathrm{LQ}(L, \{u\})$ for each word u.

We will also need the so-called right-quotient of a language with another language.

Definition 1.1.17. *Let $L_1, L_2 \subseteq \Sigma^*$ be two languages. The* right-quotient of L_1 with L_2 *is defined as* $\mathrm{RQ}(L_1, L_2) = \{u; uv \in L_1, v \in L_2\}$.

Another useful way to transform one language into a new one is by replacing symbols of its words according to some simple rule.

Definition 1.1.18. *Let Σ and Σ' be two alphabets. A* homomorphism *from Σ to Σ' is a function $\mathrm{h} : \Sigma \to (\Sigma')^*$. We extend* h *to words from Σ^* by the following rules:*

- $\mathrm{h}(\lambda) = \lambda$, *and*
- $\mathrm{h}(a_1 \dots a_n) = \mathrm{h}(a_1) \cdot \ldots \cdot \mathrm{h}(a_n)$ *for $n > 0$.*

We further extend h *to languages as follows. Let L be a language over Σ. We define* $\mathrm{h}(L) = \{\mathrm{h}(w); w \in L\}$.

The inverse homomorphism *of a language L over Σ' is defined as* $\mathrm{h}^{-1}(L) = \{w \in \Sigma^*; \mathrm{h}(w) \in L\}$.

For each regular language L there is a special DFSA, called a canonical acceptor for L, such that it accepts L and it has the minimum possible number of states among DFSA's accepting L. There is an algorithm converting each DFSA A into the canonical acceptor for $\mathrm{L}(A)$ (see [1]). It can be defined as follows.

Definition 1.1.19 ([1]). *Let $L \in \Sigma^*$ be a regular language. The* canonical acceptor for L *is defined as* $\mathrm{A}(L) = (Q, \Sigma, \delta, I, F)$ *where:*

- $Q = \{\mathrm{LQ}(L, u); u \in \mathrm{Pr}(L)\}$,
- $I = \{\mathrm{LQ}(L, \lambda)\}$ *if* L *is nonempty,* $I = \emptyset$ *otherwise,*
- $F = \{\mathrm{LQ}(L, w); w \in L\}$, *and*
- $\delta(\mathrm{LQ}(L, u), a) = \mathrm{LQ}(L, ua)$ *for* $u, ua \in \mathrm{Pr}(L)$.

We often need to combine several regular languages to obtain a new language. It is useful to know the following closure properties of the class of regular languages.

Theorem 1.1.20 ([36, 74]). *The class of regular languages is closed under union, intersection, complement, difference, concatenation, Kleene closure, left-quotient, reverse, homomorphism, and inverse homomorphism.*

There is also another way of representing a regular language — the so-called regular expressions [36]. They often provide easy to understand descriptions of regular languages.

Definition 1.1.21 ([36]). *We define regular expressions over an alphabet Σ and languages they represent as follows (we denote $\mathrm{L}(e)$ the language represented by a regular expression e):*

- $\emptyset$ *is a regular expression representing the empty language* $\emptyset$.
- λ *is a regular expression representing the language* $\{\lambda\}$.
- *For each $a \in \Sigma$ the expression a is a regular expression representing the language* $\{a\}$.
- *Let e and f be two regular expressions. Then $e + f$ is a regular expression representing the language* $\mathrm{L}(e) \cup \mathrm{L}(f)$.
- *Let e and f be two regular expressions. Then $e \cdot f$ (or simply ef) is a regular expression representing the language* $\mathrm{L}(e) \cdot \mathrm{L}(f)$.
- *Let e be a regular expression. Then e^* is a regular expression representing the language* $(\mathrm{L}(e))^*$.
- *Let e be a regular expression. Then (e) is a regular expression representing the language* $\mathrm{L}(e)$.

For example, the regular expression $0(00)^*1$ represents the language of words over $\{0, 1\}$ starting with an odd number of zeros and ending with exactly one symbol 1. In other words, $\mathrm{L}(0(00)^*1) = \{0^{2k+1}1; k \geq 0\}$.

For every finite language S, there is a very simple finite state automaton accepting exactly S. It is called a prefix tree acceptor for S and it has a tree-like structure. It is defined formally below.

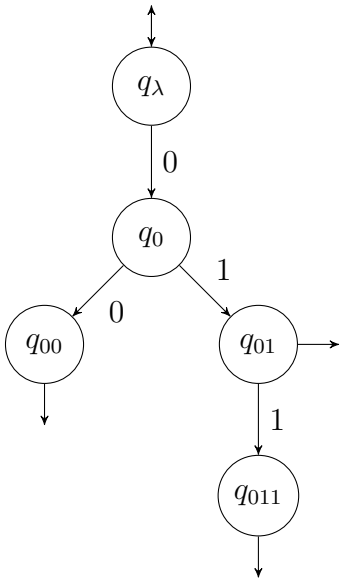


Figure 1.1: A PTA for $\{\lambda, 00, 01, 011\}$.

Definition 1.1.22. *Let $S \subseteq_{fin} \Sigma^*$. The* prefix tree acceptor *(PTA) for S is defined as $\mathrm{PTA}(S) = (Q, \Sigma, \delta, I, F)$, where*

- $Q = \mathrm{Pr}(S)$ *(i.e. the set of all the prefixes of words from S),*
- $\delta(u, a) = ua$ *for all $u, ua \in Q, u \in \Sigma^*, a \in \Sigma$,*
- $I = \{\lambda\}$ *if $S \neq \emptyset$, otherwise $I = \emptyset$, and*
- $F = S$.

The prefix tree acceptor for S can be seen as a tree where each path from the root to an accepting state corresponds to a word from the given set S. You can see an example of a PTA for $\{\lambda, 00, 01, 011\}$ in Fig. 1.1. Obviously, $\mathrm{L}(\mathrm{PTA}(S)) = S$.

There is a special technique called state merging that is used to transform FSA's in a very special way. You will see its use later. It is closely related to the so-called quotient automata introduced below.

Definition 1.1.23. *Let $A = (Q, \Sigma, \delta, I, F)$ be a FSA. Let π be a partition of Q. We define the FSA A/π called the* quotient of A and π, *where $A/\pi = (Q', \Sigma, \delta', I', F')$ and:*

- *Q' is the set of blocks of π,*
- *$\delta'(B_1, a) = \{B_2 \in \pi; (\exists b_1 \in B_1)(\exists b_2 \in B_2)(b_2 \in \delta(b_1, a))\}$, where $B_1 \in Q', a \in \Sigma$,*
- *I' is the set of all the blocks of π containing any element of I, and*
- *F' is the set of all the blocks of π containing any element of F.*

Later, we will need FSA's obtained by reverting the mechanics of another ones. This idea is formalized below:

Definition 1.1.24. *Let $A = (Q, \Sigma, \delta, I, F)$ be a FSA. The* reverse *of the transition function δ, denoted δ^{R}, is defined by $\delta^{\mathrm{R}}(q, a) = \{q' \in Q; q \in \delta(q', a)\}$ for all $a \in \Sigma$, $q \in Q$. The* reverse of the FSA *A is $A^{\mathrm{R}} = (Q, \Sigma, \delta^{\mathrm{R}}, F, I)$.*

There are several theorems characterizing the languages accepted by FSA's. One of them is the so-called pumping lemma.

Theorem 1.1.25. *Let $A = (Q, \Sigma, \delta, I, F)$ be a FSA. Then there is n such that for every word $w \in \mathrm{L}(A)$ such that $|w| \geq n$, there are $x, y, z \in \Sigma^*$ such that:*

- *$w = xyz$,*
- *$y \neq \lambda$,*
- *$|xy| \leq n$, and*
- *for all $k \geq 0$ it holds $xy^k z \in \mathrm{L}(A)$.*

Thus, having a sufficiently long word accepted by some FSA, we can be sure there is a subword that can be repeated or deleted and this way we obtain infinitely many new words accepted by that FSA.

Another useful theorem is Nerode's theorem [65]. It can be used e.g. to prove that some language is not regular or to work with the structure of a regular language.

Theorem 1.1.26 ([65])**.** *Let Σ be an alphabet and let $L \subseteq \Sigma^*$ be a language. The following conditions are equivalent:*

- *L is a regular language.*
- *There is an equivalence relation $\sim$ on Σ^* such that:*
  - *The equivalence $\sim$ is a right congruence.*
  - *The equivalence $\sim$ has a finite index.*
  - *L can be obtained as a union of some classes of $\sim$.*

Grammars are another way to represent languages (see e.g. [71, 36, 17] for more details). A grammar is a set of rules describing how to obtain all the words of the represented language:

Definition 1.1.27. *A* grammar *G is a quadruple (Σ, Γ, S, P) where Σ is a finite set of symbols called terminals, Γ is a finite set of symbols called non-terminals (where $\Sigma \cap \Gamma = \emptyset$), $S \in \Gamma$ is an initial non-terminal symbol, and P is a finite set of rules $\alpha \to \beta$, where $\alpha \in (\Gamma \cup \Sigma)^+$, α contains at least one non-terminal symbol, and $\beta \in (\Gamma \cup \Sigma)^*$.*

We write $u\alpha v \Rightarrow u\beta v$ if there is the rule $\alpha \to \beta \in P$ for all $u, v \in (\Gamma \cup \Sigma)^*$. By $\Rightarrow^*$ we denote the reflexive and transitive closure of the relation $\Rightarrow$.

The language generated by the grammar G is $\mathrm{L}(G) = \{w \in \Sigma^*; S \Rightarrow^* w\}$. For $w \in \mathrm{L}(G)$, a sequence of rewritings $S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$ which rewrites S into $w_n = w$ is called the derivation of w.

We define the following restricted versions of grammars:

- *We call G a context-sensitive grammar if all rules in P have the form $\alpha X \beta \to \alpha \gamma \beta$, where $X \in \Gamma, \alpha, \beta \in (\Gamma \cup \Sigma)^*$, and $\gamma \in (\Gamma \cup \Sigma)^+$ with the exception of $S \to \lambda$ that may occur in P only if S is not present on the right-hand side of any rule in P.*

- *We call G a context-free grammar if all rules in P have the form $X \to w$, where $X \in \Gamma, w \in (\Sigma \cup \Gamma)^*$.*

- *We call G a regular grammar if all rules in P have the form $X \to wY$ or $X \to w$, where $X, Y \in \Gamma$ and $w \in \Sigma^*$.*

- *We call G a linear grammar if all rules in P have the form $X \to uYv$ or $X \to w$, where $X, Y \in \Gamma$ and $u, v, w \in \Sigma^*$.*

- *We call G a growing context-sensitive grammar if S does not appear on the right-hand side of any rule and for each $\alpha \to \beta \in P$, where $\alpha \neq S$, it holds $|\alpha| < |\beta|$.*

A language L is context-sensitive, context-free, linear, or growing context-sensitive if there is a context-sensitive, context-free, linear, or growing context-sensitive grammar generating L, respectively. We denote the corresponding classes of languages as CSL, CFL, LIN, *and* GCSL, *respectively.*

It is a well-known result [36] that the class of languages generated by regular grammars corresponds to the already defined class of regular languages.

Automata and grammars are basic tools for representing languages. We also presented several other well known terms from the theory of formal languages. We will build upon them in the rest of this work.

1.2 Miscellaneous Basics

Relations between members of a set are often represented by graphs. This section presents some standard terms from the theory of graphs [71, 36].

Definition 1.2.1. *A* graph *is an ordered pair $G = (V, E)$, where*

- *$V = \{v_1, \ldots, v_n\}$ is a set of* vertices, *and*

- *$E \subseteq \{\{u, v\} | u, v \in V\}$ is a set of edges.*

There is a special kind of graphs having a tree-like structure that is often used when working with sentences of a natural language or when describing the run of an algorithm.

Definition 1.2.2. *We define a* tree *as follows:*

- *A graph containing only one vertex is a tree. The only vertex is called the root of the tree.*

- *Let t be a vertex and let $T_1, \ldots, T_n$ be trees (let $T_i = (V_i, E_i)$ be a tree having a root t_i (where $1 \leq i \leq n$); let $V_i \cap V_j = \emptyset$ for $i \neq j$). Then the graph $T = (\bigcup_{1 \leq i \leq n} V_i \cup \{t\}, \bigcup_{1 \leq i \leq n} (E_i \cup \{(t, t_i)\}))$ is a tree with t being its root. We call t the parent of t_i's and similarly t_i's are called the children of t.*

The vertices having no children are called leaves. *The other vertices are called* interior vertices.

For a context-free grammar G, a tree can be used to represent a derivation of some particular word using G.

Definition 1.2.3 ([36])**.** *Let $G = (\Sigma, \Gamma, S, P)$ be a context-free grammar and let $w \in \mathrm{L}(G)$. A* parse tree *for w is a tree such that:*

- *interior vertices are labeled with symbols from Γ,*
- *the root of the tree is labeled with S,*
- *leaves are labeled with either λ or a symbol from Σ,*
- *leaves labeled with λ are the only children of their parent,*
- *for each parent vertex X having children labeled with $X_1, \ldots, X_n$ (taken from the left) for some $n \geq 0$ and $X_i \in \Sigma \cup \Gamma$ (for $1 \leq i \leq n$), there is a rule $X \to X_1 \cdot \cdots \cdot X_n \in P$, and*
- *for each parent vertex X having the only child labeled with λ, there is a rule $X \to \lambda \in P$.*

We say that a derivation tree yields *the word obtained by concatenating symbols present in the leaves taken from the left to the right.*

Obviously, the parse tree for some word specifies how to derive it using rules of the grammar.

Sometimes it is useful to consider the so-called directed graph where the edges are ordered pairs of vertices:

Definition 1.2.4. *A* directed graph *is an ordered pair $G = (V, E)$, where*

- *$V = \{1, \ldots, n\}$ is a set of* vertices*, and*
- *$E \subseteq V \times V$ is a set of edges.*

If there is an edge (u, v) in a directed graph, we say the edge goes from u to v. We also say that the edge leaves u and enters v.

One of basic properties of a vertex in a graph is the number of edges it is a member of.

Definition 1.2.5. *Let $G = (V, E)$ be a graph and let $v \in V$ be a vertex. The number of edges containing v is called the* degree of v.

In the case of directed graphs we have to consider the ordered nature of edges.

Definition 1.2.6. *Let $G = (V, E)$ be a directed graph and let $v \in V$ be a vertex. The number of edges leaving v is called the* outdegree of v*, and the number of edges entering v is called the* indegree of v.

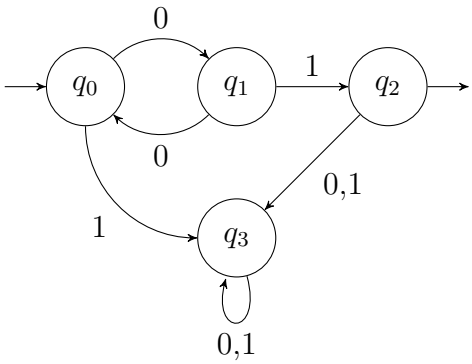


Figure 1.2: A graphical representation of a sample FSA.

Directed graphs can be used to represent the transition function in a graphical representation of a FSA. The vertices correspond to the states of the FSA, and the edges are labeled with symbols of the alphabet so that there is a transition from a state p to a state q using a symbol a whenever there is an edge going from p to q labeled with a. In order to identify the initial states, some nodes are marked with an additional arrow (with no label) entering such states. Similarly, the accepting states are denoted with an additional arrow (with no label) leaving them. In Fig. 1.2 you can see a graphical representation of the DFSA $A = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, \{q_0\}, \{q_2\})$, where:

- $\delta(q_0, 0) = q_1$,
- $\delta(q_0, 1) = q_3$,
- $\delta(q_1, 0) = q_0$,
- $\delta(q_1, 1) = q_2$,
- $\delta(q_2, 0) = q_3$,
- $\delta(q_2, 1) = q_3$,
- $\delta(q_3, 0) = q_3$, and
- $\delta(q_3, 1) = q_3$.

Obviously, $\mathrm{L}(A) = \{0^{2k+1}1; k \geq 0\}$. The state q_3 is a dead state.

2. Related Work

This chapter presents several approaches to the *grammatical inference* problem, briefly stated in the Introduction. We will discuss learning scenarios, data sources, etc., this time being more formal.

In Section 2.1 we present basic results from the field of grammatical inference, such as some approaches to the learnability of languages, etc. In Section 2.2 we introduce the ABR approach and restarting automata.

2.1 Grammatical Inference

This section introduces the grammatical inference in general. Several approaches to the learning problem are presented together with well-known results that not only offer basics to build on, but also illustrate the barriers present in this field. For a more comprehensive review, see e.g. [27].

Learning of some unknown language, called *a target language,* is often based on some sample words both from the language and from its complement (see e.g. [23, 4, 18]). The process of learning a model of an unknown target language from given samples is called *grammatical inference*. The following definition formalizes the concept of the mentioned samples (there are several approaches to the input samples as you will see later).

Definition 2.1.1. *Let L be a language over Σ. A* set of positive samples of the language L *is a finite subset of L. A* set of negative samples of the language L *is a finite subset of $\Sigma^* \setminus L$. A* sample of the language L *is a pair (S^+, S^-) where S^+ is a set of positive samples of the language L and S^- is a set of negative samples of the language L.*

The goal in grammatical inference is to use the given sample of the unknown target language L in order to find a hypothesis language L' that is as close to the unknown target language L as possible. Having a sample (S^+, S^-) of an unknown target language L and a hypothesis language L', we may ask if L' is a reasonable candidate for the language L. By this we mean if the hypothesis L' is consistent with the sample (S^+, S^-). This idea is formally defined below.

Definition 2.1.2. *Let (S^+, S^-) be a sample of a language L. We say that a language L' is* consistent *with (S^+, S^-) if $S^+ \subseteq L'$ and $S^- \cap L' = \emptyset$.*

Below we try to define the inference problem very informally. It is just an attempt to define an ideal problem. However, the output of the problem is ill-defined as discussed below. This definition will be later justified to fit our particular needs and to reach the needed level of formality and correctness.

Definition 2.1.3. *Let L be an unknown target language and let (S^+, S^-) be a sample of L. The problem of finding the language L using the sample (S^+, S^-) is called an* inference problem. *An algorithm for solving the inference problem is called an* inference algorithm.

The problem with this definition is well illustrated by the following example. Let us have the following unknown target languages:

- $L_0 = \{w \in \{0,1\}^*; |w| \text{ is even}\}$, and
- $L_1 = \{w \in \{0,1\}^*; w \text{ contains at least one 0 and at least one 1}\}$.

Then the sample (S^+, S^-) where

- $S^+ = \{01, 0111, 111100\}$, and
- $S^- = \{0, 111\}$

can be supplied to an inference algorithm with the intent to obtain both L_0 and L_1. Thus, we would like our inference algorithm to return two different results on the same input data. Therefore, the problem is unsolvable. Thus, a modification of this approach is needed for the definition of the problem to be of practical use. We will present several approaches by other researchers below in this section. The purpose of the simple definition above was to give you an idea of what is going on — that the goal is to learn some language from given samples.

Gold [24] proposed a basic model of grammatical inference where the learner iteratively improves his hypothesis as more and more input samples are supplied by a teacher. The input samples are just words labeled as either a positive or a negative sample. They are supplied in a form of an infinite sequence, called a presentation of a language.

Definition 2.1.4 ([24]). *A presentation of a language $L \subseteq \Sigma^*$ is an infinite sequence of pairs $\{(w_i, s_i)\}_{i=0}^{\infty}$ such that*

- $\{w_i; i \in \mathbb{N}\} = \Sigma^*$, *i.e. no sample is omitted,*
- $s_i = 0$ *for* $w_i \notin L$, *and*
- $s_i = 1$ *for* $w_i \in L$.

After obtaining a new input sample, the learner guesses a new hypothesis. We say that the target language was *identified in the limit* if from some time on, the hypothesis represents the target language and remains the same forever.

Definition 2.1.5 ([24]). *Let Σ be an alphabet and let A be an algorithm that takes a sequence of labeled words $\{(w_i, s_i)\}_{i=0}^{n}$ (where $w_i \in \Sigma^*$ is a word, $s_i \in \{0,1\}$ is a label, and $n \in \mathbb{N}$) and then it outputs an automaton M_n representing a model of the n-th hypothesis. The algorithm A* identifies the language $L \subseteq \Sigma^*$ in the limit *if for any presentation $\{(w_i, s_i)\}_{i=0}^{\infty}$ of L there is n_0 such that for all $n \geq n_0$ the algorithm A applied onto $\{(w_i, s_i)\}_{i=0}^{n}$ returns M_n such that the language accepted by M_n is equal to L. A class of languages $\mathcal{L}$ is* learnable in the limit *if there is an algorithm A such that for all $L \in \mathcal{L}$ it holds that A identifies the language L in the limit.*

In other words, an algorithm identifies a language in the limit if it converges to a correct hypothesis regardless of the actual presentation of the target language as long as no sample is omitted.

A basic result regarding this approach is that the class of primitive recursive languages, that is a superclass of context-sensitive languages, is learnable in the limit [24]. This is a quite large class of languages but note that during the

inference process the learner can never be sure that the target language was already identified.

The simplest class of languages in well-known Chomsky hierarchy [11] is the class of languages that can be represented by FSA's. This class was thoroughly investigated, and a number of interesting results regarding inference of FSA's emerged, revealing the complexity of the inference task even in this simple case. We know that regular languages can be identified in the limit [23]. But more importantly — are we able to identify them in practice? Unfortunately, even to find the smallest DFSA consistent with given labeled words is known to be an NP-hard problem [23].

Trakhtenbrot and Barzdin [69] showed that the problem is tractable when all words up to the length equal to the size of the target are presented to the learner. However, note the enormous number of input samples to be supplied to the learner. Angluin [7] studied the case when a very small fraction of those words is missing in the input, and she proved that the problem remains NP-hard. Another attempt to make the problem easier was performed by Pitt and Warmuth [63]. They allowed a slightly larger resulting automata — they consider the problem of finding a DFSA that is only polynomially larger than the minimum DFSA (i.e. the DFSA having the minimum number of states among all consistent automata), but still consistent with the input samples. Unfortunately, even this problem turned to be NP-hard. Another point of view was introduced by Kearns and Valiant [41] who showed that learning of DFSA's is related to solving hard cryptographic problems.

We may quite naturally ask why to bother with searching for small automata. In practice, the input data are often too small to identify the target exactly. Even worse, there is usually infinitely many possible models consistent with given input samples. The most important criterion can be that the resulting model should be consistent with unseen samples that may occur in the future (e.g. in the case of learning a language in order to classify unseen sample later). Unfortunately, this criterion is hard to evaluate exactly. One well-known idea for dealing with such problems is the so-called Occam's razor [54]. It states that a simple model is better than a complex one. In the case of DFSA's, we can consider the length of the description of the model to be the complexity measure. It can be e.g. equal to the number of states of a canonical acceptor for the guessed language or computed using the so-called minimal message length [28] approach. This leads directly to the idea of searching for small DFSA's.

Above we summarized several negative results regarding learning of languages. Unfortunately, the learning task is often even harder. In contrast to learning from samples labeled as positive or negative, we are often presented only with the positive ones. Those samples can represent e.g. our observations of how something works, some samples of things we would like to identify among others, etc. It is easy to see (as shown by Gold [23]) that if only words of an unknown target language are supplied to the learner, then it is impossible to identify any class containing all finite languages and at least one infinite one.

Though it could seem necessary to supply negative input samples to learn some more complex languages, it is not the case. Note that limiting the power of considered models could help. Let us suppose we have an idea of learning some interesting language L. Let us further suppose that this language is not regular.

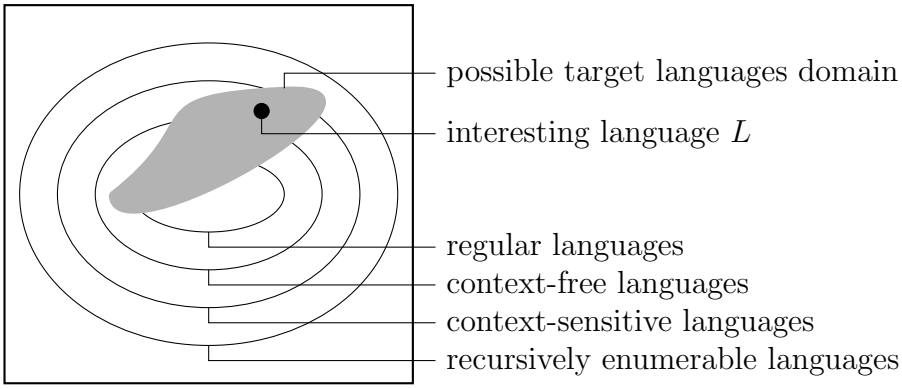


Figure 2.1: By limiting ourselves onto a reasonable class of languages, we can still learn interesting languages from positive data only.

Does it mean that it cannot be inferred using positive samples because of the Gold's result? No — it suffices to limit reasonably the class of languages being considered by the learning algorithm so that it still contains the language L, but on the other hand, it becomes learnable in the limit from positive data. This idea is depicted in Fig. 2.1.

For example, Angluin [4] restricted the class of regular languages by introducing its subclass called reversible languages. She also proposed an algorithm for learning them. Similarly, García and Vidal [22] proposed an algorithm for learning the so-called k-testable languages.

Many attempts to learn regular languages are based on FSA's and a merging technique (see e.g. Miclet [53], Angluin [4], Dupont [20], and Hoffmann [30]). Let (S^+, S^-) be a sample of an unknown *regular* language L consisting of a set of positive samples $S^+ \subseteq L$ and a set of negative samples S^- such that $S^- \cap L = \emptyset$. Are we able to guess the language L based on given samples (S^+, S^-)? The problem of *regular inference from positive and negative samples* is to find a *regular* language L' consistent with input samples (i.e. such that $S^+ \subseteq L'$ and $S^- \cap L' = \emptyset$), and to try to approximate L by L' as well as can be. Note the similarity to the Definition 2.1.3. We know that for representing regular languages we can use DFSA's. Therefore, the problem of regular inference from positive and negative samples can be seen as a search for a corresponding DFSA A consistent with (S^+, S^-) (by this we mean that L(A) is consistent with the input samples) and approximating the unknown target language. As already mentioned, there can be infinitely many possible candidates for the hypothesis language L', and there is a reason for preferring simple hypotheses to more complex ones. By merging states of a given FSA, we obtain a quotient automaton of that FSA and some partition of its states. Thus, the effect of using the merging technique is also lowering the number of states of the particular FSA. It can be therefore considered to be a tool that can be used to reach small DFSA's.

While merging, it is natural to allow only hypotheses A (FSA's, i.e. we consider NFSA's as members of the search space, because merging can introduce nondeterminism) such that [20, 28]:

(H1) A accepts all samples from S^+, and

(H2) every transition of A and every accepting state of A are used to accept at least one sample from S^+.

As every quotient automaton of PTA(S^+) satisfies [20] both (H1) and (H2), it is possible to start with PTA(S^+) (which itself obviously belongs to the searched space) and search by merging its states without a risk of leaving the search space. Moreover, as the minimum DFSA consistent with a given sample (S^+, S^-) (that can be thought as an ideal solution) is known to be a quotient automaton of PTA(S^+) [20], we know that it is reachable by following the merging process. It is therefore natural to make partitions of the states of PTA(S^+) represent individual FSA's from the search space.

Thus, we search a lattice of all quotient automata of PTA(S^+) (where $A \preceq B$ if A is a quotient of B and some partition of states of B). In contrast to poor results achieved [43] using the depth-first search, the evidence driven technique [45] won the Abbadingo competition [44], where many regular inference algorithms competed on learning tasks of several complexity levels.

Another well known algorithm using the merging technique is RPNI (Regular Positive and Negative Inference) by Oncina and García [58] (later extended by Hoffmann [30] by introducing a simplicity measure for guiding the learning process). It also starts with the prefix tree acceptor for positive samples, and it merges its states while assuring that no negative input sample will be accepted by the resulting automaton. This algorithm returns the target DFSA in polynomial time if the input samples satisfy some additional conditions [58, 19]. We present it here as it is quite easy to understand and it can serve as an example of a learning algorithm (the algorithm taken from [61]).

We start by creating a prefix tree acceptor PTA(S^+). Its states are numbered according to the standard order of the set $\Pr(S^+)$ (shorter words precede longer ones, words of the same length are ordered lexicographically) as $0, \ldots, N-1$. Then we work in a cycle trying to somehow merge as much as possible, but checking if the resulting automaton remains consistent with input samples.

Algorithm 2.1.6. Algorithm RPNI.
Input: a sample (S^+, S^-) of some unknown target language L.
Output: a DFSA consistent with (S^+, S^-).

Let $M = \text{PTA}(S^+)$. Let $0, 1, \ldots, (N-1)$ denote the states of M.
Let $\pi = \pi_0 = \{\{0\}, \{1\}, \ldots, \{N-1\}\}$.
for $i = 1$ **to** $N-1$ **do**
begin
  for $j = 0$ **to** $i-1$ **do**
  begin
    Let $\pi' = (\pi \setminus \{[i]_\pi, [j]_\pi\}) \cup \{[i]_\pi \cup [j]_\pi\}$.
    Let $M_{\pi'} = M/\pi'$.
    Let $M_{\pi''} = \text{deterministicMerge}(M_{\pi'})$.
    if consistent$(M_{\pi''}, S^-)$ **then**
    begin
      Let $M = M_{\pi''}$.
      Let $\pi = \pi''$.
      break
    end

end
end
return M.

The function deterministicMerge($M_{\pi'}$) turns $M_{\pi'}$ into a DFSA by iteratively merging blocks of π' causing non-determinism until the resulting partition corresponds to a DFSA.

From the code above it is clear that the output of that algorithm is always a DFSA consistent with the input samples. This was an example algorithm inferring models of regular languages from given samples. In other words, an unknown target language is being guessed based on some given samples.

In the case of identification in the limit, we expect the presentation of the unknown target language to contain all words over the considered alphabet. For some classes of languages and corresponding models, it can be shown that there is an upper limit on the number of samples needed (if the samples are chosen appropriately) to identify the target language exactly — for example, the limit can be a polynomial in the size of the model corresponding to the target language. This leads to the following definition.

Definition 2.1.7 ([26])**.** *A class of models $\mathcal{M}$ is* identifiable in the limit from polynomial time and data *if there exist two polynomials $p()$ and $q()$ and an algorithm A such that the following holds:*

1. *Let L be a target language and let (S^+, S^-) ($S^+ \subseteq L, S^- \cap L = \emptyset$) be a sample of size m. Then the algorithm A returns a model $M \in \mathcal{M}$ consistent with (S^+, S^-) in time $O(p(m))$.*

2. *Let M be a model of size n. Then there exists a* characteristic sample *(CS^+, CS^-) such that:*

   - *the size of (CS^+, CS^-) is less than $q(n)$, and*
   - *on the input (S^+, S^-) such that $CS^+ \subseteq S^+$ and $CS^- \subseteq S^-$, the algorithm A returns a model M' equivalent to M.*

However, the ability to identify a class of models in the limit from polynomial time and data does not imply the ability to infer models from that class effectively. See e.g. the case of DFSA's in the following theorem.

Theorem 2.1.8 ([26, 23])**.** *It holds*

1. *DFSA's are identifiable in the limit from polynomial time and data.*

2. *The class of context-free grammars over Σ, when $|\Sigma| > 1$, is not identifiable in the limit from polynomial time and data.*

3. *The class of linear grammars over Σ, when $|\Sigma| > 1$, is not identifiable in the limit from polynomial time and data.*

Up to now we considered a simple learning scenario — the teacher supplies samples and the learner returns his hypotheses. It could be very useful if the learner can inform the teacher about what information he needs — e.g. in form of

a question whether some particular word belongs to the unknown target language. This could make the learning process faster. The problem setting presented below reflects this by considering the queries.

To introduce queries, the so-called oracle can be used that is able to answer queries. We can have oracles for different kinds of queries. The so-called membership queries ask whether a particular word is a member of the target language. Another kind are the so-called equivalence queries, asking whether a given grammar corresponds to the target and if it is not the case, then the oracle returns also a counter-example. Those kinds of oracles were considered by Angluin [8, 6], who proved that the learning tasks remains hard even if those queries are allowed. A different situation occurs when we allow both kinds of queries to be used at once — this extended oracle is called a Minimal Adequate Teacher (denoted MAT).

Definition 2.1.9 ([5])**.** *A minimally adequate teacher is an oracle able to answer the following queries about an unknown language L:*

- *the answer to a* membership query *for a word w is* yes *if $w \in L$ and* no *otherwise, and*
- *the answer to an* equivalence query *for a model M of a regular language is:*
  - yes *if the language represented by M (let it be L') is equal to L, or*
  - *a counterexample $w \in L \oplus L'$ otherwise.*

Angluin proposed [5] an algorithm called L* that uses a MAT to learn an unknown regular language.

Theorem 2.1.10 ([5])**.** *Let L be an unknown regular language. Let T be a MAT for L. Then the following holds:*

- *The* L* *algorithm using T returns a DFSA M such that*
  - $\mathrm{L}(M) = L$*, and*
  - *M has the minimum number of states among such automata.*
- *The algorithm runs in time bounded by a polynomial in n and d where*
  - *n is the number of states of M, and*
  - *d is an upper bound on the length of any counterexample presented by T.*

Exact learning is both hard and sometimes unnecessary. Sometimes it could be sufficient to have a model that only reasonably approximates the target. And even more, we could admit that on a small fraction of inputs our learning algorithm does not perform well. This approach was introduced by Valiant [70] as a *probably approximately correct* (PAC) model.

Definition 2.1.11 ([57])**.** *An algorithm A is a PAC learning algorithm for a class of languages $\mathcal{L}$ if the following holds:*

- *The algorithm A expects three parameters:*

- *the error parameter* $\epsilon \in (0, 1]$,
- *the confidence parameter* $\delta \in (0, 1]$, *and*
- *the length parameter* $n \in \mathbb{N}$.

- *The algorithm A obtains classified samples of some language $L \in \mathcal{L}$ by calling a function* EXAMPLE. *The samples are selected at random according to a probability distribution P defined over the set of all words up to the length of n.*
- *For all $L \in \mathcal{L}$ and all distributions P, algorithm A returns a language $L' \in \mathcal{L}$ such that with probability at least $(1 - \delta)$ it holds $P(L \oplus L') \leq \epsilon$, where $P(S) = \Sigma_{w \in S} P(w)$ and $P(w)$ denotes the probability that the word w will be selected according to the distribution P.*

We expect that the iterative approach to parsing words inherent in the main model considered in this thesis makes it difficult to prove results on more advanced approaches to grammatical inference presented in this section. Therefore, we decided to consider an approach that is similar to Gold's one [24] presented above. The algorithm we propose in the following text can be used iteratively. At each step, the teacher supplies some new sample to possibly improve the performance of the resulting automaton. However, instead of learning from bare words, we will consider richer input samples.

2.2 Analysis by Reduction Inference

This section introduces the ABR approach and its relation to the so-called restarting automata.

When learning a natural language, any information about the underlying structure of sentences can help. For example, in the case of English, it would be helpful to know that sentences often have the form of "subject — verb — object — manner — place — time". Another example of such additional information is a reduction relation — the sentence "Peter works slowly." is an extended version of the sentence "Petr works." In this thesis, the reduction relation plays an important role. It gives us a tool for iterative parsing of given words. An example iterative parsing of a word is given below (remember here English words are called symbols and the whole sentences are called words; the parts of the sample words modified in individual steps are underlined):

<u>Peter and Jane</u> work very slowly.
$\Downarrow$
They work <u>very</u> slowly.
$\Downarrow$
They work <u>slowly</u>.
$\Downarrow$
They work.

We performed three steps (the so-called reductions), each time replacing a subword of the current word with a shorter one — at first we replaced "Peter

and Jane" with "They", then we replaced the symbol "very" with the empty word (in other words, we deleted the symbol "very"), and finally we removed the symbol "slowly". Because the word "They work." is syntactically correct and our reduction relation preserves errors, we conclude that the initial word "Peter and Jane work very slowly." is correct as well. Note that we do not specify the reduction relation here, but for this example we supposed some natural sense of what a reduction relation for English could look like. It is natural to require that errors are not removed when checking the original words for correctness. It would otherwise break the whole parsing process — the words containing errors would be considered right. This way of parsing is called the *analysis by reduction* (ABR for short) [37, 49, 64].

The ABR is a useful way of parsing sentences from natural languages. As illustrated above, it consists in stepwise simplifications of a given extended sentence until a correct simple sentence is obtained or an error is found. We will sometimes use the term ABR also to denote a particular sequence of sentences obtained while parsing a sentence by the ABR method.

To model the ABR, one can use the so-called restarting automata [39, 60, 59]. They are the core concept of this thesis. We will therefore present them formally now together with relevant results.

Definition 2.2.1 ([39, 60, 59])**.** *A* restarting automaton *(*RRWW*-automaton) is a triplet $M = (\Sigma, \Gamma, I)$, where*

- Σ *is an input alphabet,*
- Γ *is a working alphabet,* $\Sigma \subseteq \Gamma$ *(the symbols from* $\Gamma \setminus \Sigma$ *are called auxiliary symbols), and*
- I *is a finite set of meta-instructions of the following two forms:*
  - *a rewriting meta-instruction* $(E_\ell, x \to y, E_\mathrm{r})$*, where* $x, y \in \Gamma^*$ *such that* $|x| > |y|$ *and* $E_\ell, E_\mathrm{r} \subseteq \Gamma^*$ *are regular languages (we call the expression* $x \to y$ *a* rewrite*, the words* x *and* y *are called a* rewritten *and a* replacement *word, respectively, and* $|x|$ *is called the* length of the rewrite*), or*
  - *an accepting meta-instruction* (E, Accept)*, where* $E \subseteq \Gamma^*$ *is a regular language.*

The maximum length of a rewrite taken over all rewriting meta-instructions is called the size of the window of M*.*

The language obtained as a union of languages present in accepting meta-instructions is called the language of simple words or the base language of M*.*

We say that M *can* directly reduce *a word* u *into a word* v *(*$u \vdash_M v$*) if* $u = u_1 x u_2, v = u_1 y u_2$ *for some* $u_1, u_2, x, y \in \Gamma^*$ *and there is a rewriting meta-instruction* $(E_\ell, x \to y, E_\mathrm{r}) \in I$*, where* $u_1 \in E_\ell$ *and* $u_2 \in E_\mathrm{r}$*. The action of replacing* x *with* y *in the word* $u = u_1 x u_2$ *is called a* rewriting*. Let* $\vdash_M^*$ *denote the reflexive and transitive closure of* $\vdash_M$*. If* $u \vdash_M^* v$ *for some words* u, v*, we say that* M *can* reduce u *to* v*.*

A word $w \in \Gamma^*$ is accepted by M directly *if $w \in E$ for some accepting meta-instruction $(E, \text{Accept}) \in I$. A word $w \in \Gamma^*$ is* accepted by M *if there exists a word $w' \in \Gamma^*$ such that $w \vdash_M^* w'$ and w' is accepted by M directly.*

A characteristic language accepted by M *is the set of all words accepted by M, thus $\mathrm{L_C}(M) = \{w \in \Gamma^*; w \text{ is accepted by } M\}$.*

A language accepted by M *is the set of all words over the input alphabet Σ accepted by M, thus $\mathrm{L}(M) = \mathrm{L_C}(M) \cap \Sigma^*$.*

We define two restricted versions of RRWW*-automata:*

- *An* RRWW*-automaton having $\Gamma = \Sigma$ (i.e. an automaton using no auxiliary symbols) is called an* RRW*-automaton.*
- *An* RR*-automaton is an* RRW*-automaton such that for each rewriting meta-instruction $(E_\ell, u \to v, E_\mathrm{r})$, it holds that v can be obtained from u by deleting some symbols from u.*

By $\mathcal{A}(X)$ and by $\mathcal{L}(X)$ for $X \in \{\mathsf{RR}, \mathsf{RRW}, \mathsf{RRWW}\}$ we denote the class of automata of type X and the class of languages accepted by automata of type X, respectively.

Fig. 2.2 illustrates an accepting computation of the RR-automaton $M = (\{0,1\}, \{0,1\}, \{I_1, I_2\})$, where

- $I_1 = (L_\ell = \{0^i; i \geq 0\}, 01 \to \lambda, L_\mathrm{r} = \{1^i; i \geq 0\})$ is a rewriting meta-instruction, and
- $I_2 = (L_\mathrm{b} = \{01\}, \text{Accept})$ is an accepting meta-instruction.

This automaton accepts the language $\{0^n 1^n; n \geq 1\}$. You can think of an RRWW-automaton as a device having a finite control unit (containing the meta-instructions) and a head attached to a tape containing the input word. It works in phases. During each phase, the automaton nondeterministically selects one of its meta-instructions i and scans the content of its tape. According to the word on its tape, it either halts and accepts, or halts and rejects, or it performs a reduction and starts a new phase. Let i be an accepting meta-instruction of the form $i = (E, \text{Accept})$. Then if the current word on its tape belongs to E, M halts and accepts, otherwise it halts and rejects. If i is a rewriting meta-instruction of the form $(E_\ell, x \to y, E_\mathrm{r})$, then if the current tape content is of the form uxv for some words $u \in E_\ell$ and $v \in E_\mathrm{r}$, then M rewrites x by y and restarts, i.e. the control unit resets and a new phase starts on the shortened word. Once a word is accepted, the initial input word is accepted as well. So, in our example, the initial input word 000111 is first reduced to 0011 by replacing its subword 01 with the empty word λ using the rewriting meta-instruction I_1. Then the word 0011 is reduced again using the same meta-instruction to get the word 01. This word is then accepted directly using the accepting meta-instruction I_2. We conclude that the initial input word 000111 is accepted as well.

We will often use regular expressions to describe languages present in meta-instructions of restarting automata. For simplicity, we will write just the regular expressions such as 0^* instead of more formally correct $\mathrm{L}(0^*)$ to specify the language represented by the regular expression.

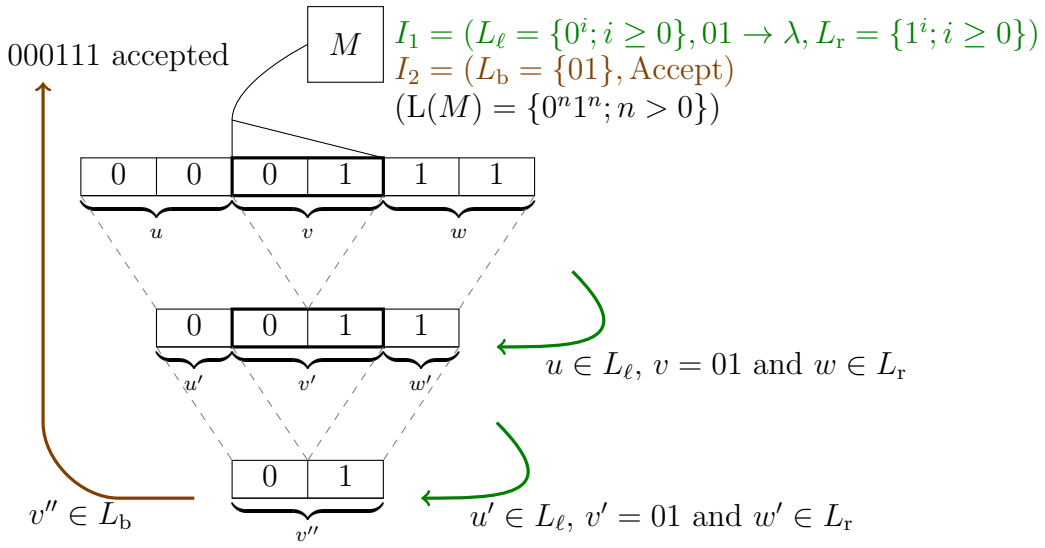


Figure 2.2: A sample computation of an RR-automaton.

[59] is a great overview of different variants of restarting automata which presents their basic properties and studies the relations among the respective language classes and the Chomsky hierarchy. One of the fundamental properties of all models of restarting automata is the following so-called error preserving property. This property means that whatever reduction performs given restarting automaton on an input word not belonging to the language accepted by the automaton ("the word contains an error"), the resulting word is not accepted by the automaton (it contains still some error), too.

Proposition 2.2.2 (Error Preserving Property; [37]). *Let $M = (\Sigma, \Gamma, I)$ be an RRWW-automaton, and let u, v be words over its input alphabet Σ. If $u \vdash_M^* v$ holds and $u \notin \mathrm{L}(M)$, then $v \notin \mathrm{L}(M)$, either.*

This proposition is extensively used for proving that some language can not be accepted by any restarting automaton of a certain type (see [59]).

For further results about restarting automata see [59].

It would be very helpful to apply restarting automata for solving practical problems. However, it is not easy to design restarting automata by hand for languages as complex as the natural ones. Therefore, a method of inferring restarting automata from samples is needed. As already mentioned in Introduction, there were already some attempts to learn restarting automata using genetic algorithms [12, 13, 32]. However, the achieved results are far from being applicable. A method of learning restarting automata from positive samples only was proposed in [55]. The paper proposed a subclass of restarting automata, the so-called strictly locally testable restarting automata, together with a protocol for learning them. The results from [55] are closely related to the method proposed in this thesis. We will comment this relation several times in the following text.

A similar approach to grammatical inference based on rewriting was proposed also in [21] where a special kind of string rewriting systems was considered. Instead of speaking about strings, we will use the term "words" to stay consistent with the rest of this thesis. The system is given using some rewriting rules and an irreducible word. Then a given word belongs to the represented language if

the word can be rewritten to the irreducible word using the specified rules. In general, a rewriting rule $l \vdash r$ says that a subword l can be replaced with the word r. In the mentioned paper, the general rewriting systems are slightly modified to increase the power of the model, and several conditions on the set of rules are considered to ensure both finiteness of derivations and the Church-Rosser property of rewriting systems (the resulting model is called a hybrid almost non-overlapping delimited string-rewriting system). The proposed class of rewriting systems is powerful enough to represent all regular languages and several typical context-free languages. Moreover, a learning algorithm called LARS (Learning Algorithm for Rewriting Systems) is provided that is able to learn systems representing a subclass of these languages from given sample words of the unknown target language and given sample words of its complement. We compare our method with LARS in Subsection 7.3.1.

As we mentioned already, the ABR iteratively simplifies the given word. Having only a sequence of words obtained during the ABR, it may not be always clear what actually happened if e.g. 0101 is directly reduced to 01. There are several options:

- $\underline{01}01 \Rightarrow 01$ (the prefix 01 is removed),
- $0\underline{10}1 \Rightarrow 01$ (the subword 10 is removed),
- $01\underline{01} \Rightarrow 01$ (the suffix 01 is removed),
- $\underline{010}1 \Rightarrow 01$ (the prefix 010 is replaced with 0),
- $0\underline{101} \Rightarrow 01$ (the suffix 101 is replaced with 1), or
- $\underline{0101} \Rightarrow 01$ (the whole word 0101 is replaced with 01).

When describing the rules for performing the ABR, the exact location of changes in individual reductions plays an important role. Therefore, in this thesis, we distinguish the case when the exact location of a change is given and when the exact location is unspecified.

Definition 2.2.3. *The* set of all possible located reductions *over an alphabet* Γ *is the set* $\dot{\mathrm{R}}(\Gamma) = \{(u, x \to y, v); (u, v, x, y \in \Gamma^*)$ *and* $(|x| > |y|)\}$.

The set of all possible sliding reductions *over an alphabet* Γ *is the set* $\overline{\mathrm{R}}(\Gamma) = \{(w \Rightarrow w'); (w, w' \in \Gamma^*)$ *and* $(|w| > |w'|)\}$.

For a located reduction $(u, x \to y, v)$*, we call the expression* $x \to y$ *a* rewrite*, the words* x *and* y *are called a* rewritten *and a* replacement *word, respectively,* $|x|$ *is called the* length of the rewrite*, and the action of replacing* x *with* y *in the word* uxv *is called a* rewriting*.*

For a located reduction $(u, x \to y, v)$ *and a sliding reduction* $(uxv \Rightarrow uyv)$*, we say that they* reduce *the word* uxv *to the word* uyv*.*

Please note that a rewriting meta-instruction corresponds to a subset of the set of all possible located reductions. Hence, it is natural to write $(u, x \to y, v) \in (E_\ell, x \to y, E_\mathrm{r})$ when $u \in E_\ell$ and $v \in E_\mathrm{r}$.

Similarly, in the case of sliding reductions, we write $(w \Rightarrow w') \in (E_\ell, x \to y, E_\mathrm{r})$ if there are $u, v \in \Gamma^*$ such that $w = uxv$, $w' = uyv$, $u \in E_\ell$, and $v \in E_\mathrm{r}$.

In Definition 2.1.1 we used sets of positive and negative sample words to learn a model of a target language. Now, we would like to supply additional information about the reduction relation. We therefore define the input as follows:

Definition 2.2.4. *Let* Γ *be an alphabet. Let* $R \in \{\dot{\mathrm{R}}, \overline{\mathrm{R}}\}$. *An* R-presentation of a restarting automaton *over* Γ *is a quadruple* $S = (S^+, S^-, R^+, R^-)$, *where*

- $S^+ \subseteq \Gamma^*$ *is a finite set of* positive samples,
- $S^- \subseteq \Gamma^*$ *is a finite set of* negative samples,
- $R^+ \subseteq R(\Gamma)$ *is a finite set of* positive sample reductions, *and*
- $R^- \subseteq R(\Gamma)$ *is a finite set of* negative sample reductions.

We call the R*-presentation* positive *if* $S^- = R^- = \emptyset$. *Otherwise we call it positive and negative.*

A positive and negative $\overline{\mathrm{R}}$-presentation of a restarting automaton accepting a subset of English could be given for example as $S = (S^+, S^-, R^+, R^-)$, where (individual samples are separated by a semicolon):

- $S^+ = \{$Peter and Jane work very slowly.; Peter works.$\}$,
- $S^- = \{$Work Peter.$\}$,
- $R^+ = \{$Peter works very slowly. $\Rightarrow$ Peter works slowly.$\}$, and
- $R^- = \{\}$.

.

Of course, this sample is probably not enough to learn English, but it is a very basic example of an $\overline{\mathrm{R}}$-presentation.

Note that no relation between an R-presentation and any particular restarting automaton is needed. We can define some R-presentation without having any restarting automaton at hand.

If we use a presentation to infer a restarting automaton, it is natural to ask whether the obtained automaton is somehow consistent with the input presentation. For the purpose of this thesis, we consider the following definition of consistency.

Definition 2.2.5. *Let* $R \in \{\dot{\mathrm{R}}, \overline{\mathrm{R}}\}$. *We say that a restarting automaton* $M = (\Sigma, \Gamma, I)$ *is consistent with* R*-presentation* $S = (S^+, S^-, R^+, R^-)$, *if*

- *for each* $s \in S^+$ *it holds* $s \in \mathrm{L}(M)$,
- *for each* $s \in S^-$ *it holds* $s \notin \mathrm{L}(M)$,
- *for each* $r \in R^+$ *there exists a rewriting meta-instruction* $i \in I$ *such that* $r \in i$, *and*
- *for each* $r \in R^-$ *and each rewriting meta-instruction* $i \in I$ *it holds* $r \notin i$.

Thus, the automaton has to satisfy requirements given by all four elements of the presentation. In practice, if our goal is to obtain an automaton accepting some target language, we need not require that all positive reductions from R^+ must be performed by the obtained automaton if the automaton still accepts the target language. Therefore, we may then relax the definition e.g. by removing the requirement related to the set of positive reductions R^+ and consider them to be only hints to the learning algorithm. However, in this thesis we will not consider such a relaxation.

We could also consider a slightly different definition of consistency of a restarting automaton with a presentation — instead of requiring that samples from S^+ are accepted by M we could require that they are accepted by M *directly*. Similarly, samples from S^- could be checked against the accepting meta-instructions only as well. For simplicity, for the rest of this thesis, we consider only the definition as presented above (but as you will see yourself, the difference is not always relevant).

Actually, this thesis does not propose a method for learning restarting automata consistent with given presentations as defined above. This is just an initial approach to learning restarting automata, that is similar to approaches used by other researchers. We will further formalize this approach later in Chapter 4. It will then help us to understand the complexity of learning restarting automata in this way, and we will build upon it to eventually obtain a definition of the inference problem that we will try to solve.

3. Goals

In this chapter we summarize the goals of this thesis.

1. **Complexity** The problem of learning the ABR is relatively new and it is important to know its complexity. Having Occam's razor in mind, we will analyze the complexity of the problem of searching for a reasonably small models of the ABR consistent with given input samples, similarly to other researchers. The obtained results are presented in Chapter 5.

2. **Algorithm** To learn a model of the ABR from given samples (particularly from positive samples only) would be very useful in many areas. Based on the results on the complexity of learning the ABR, we will propose a new method for inferring the ABR by learning a subclass of restarting automata from given samples. Moreover, we will relate the class of languages accepted by the new model of restarting automata to Chomsky hierarchy. This will establish also limits of learnability of the ABR using our model. Our new method is proposed in Chapter 6.

3. **Benchmark** As it turned out, the problem of learning an unknown target language from positive samples only is very hard. Our proposed method tries to somehow approximate the optimal solution. It is even possible that the exact solution does not exist in the search space. A usual way to measure the quality of an inference algorithm is to run the algorithm on several sample problems and then to evaluate the outputs. Our inference algorithm requires a much richer input samples than usual inference algorithms. Therefore, it also requires a different benchmark. We will analyze some known benchmarks and propose a new one. These topics are present in Chapter 7 together with the results obtained by our proposed method.

The following chapters are dedicated to achieving the above presented goals. We start this process in the next chapter where we summarize the core definitions of problems we deal with in later chapters.

4. Problem Definition

This chapter presents several problems that are discussed in this thesis. We would like to study learning the ABR, hence we will first formally define the problem of learning the ABR. The definition is similar to definitions commonly used in the field of grammatical inference. It asks for a small model consistent with given input samples. Actually, the definition from Section 4.1 will be used for establishing the complexity of inference of the ABR. Later, in Section 6, we propose an inference algorithm for solving a slightly different problem which we define in Section 4.2.

4.1 Definition for Analysis of Complexity

The rather informal Definition 2.1.3 considers learning from sample words only. In Definition 2.2.4 we introduced additional source of information that can be used in the learning process. You probably have an idea of a corresponding extended definition of the inference problem. However, a more formal definition is needed to analyze the problem complexity later in Chapter 5.

Definition 4.1.1. *Let* $\mathsf{X} \in \{\mathsf{RRWW}, \mathsf{RRW}, \mathsf{RR}\}$ *be a type of restarting automata, let* $\alpha : \mathcal{A}(\mathsf{X}) \to \mathbb{N}$ *be a function mapping X-automata into non-negative integers, let* $k \in \mathbb{N} \cup \{\infty\}$*, let* Σ *and* Γ *be arbitrary alphabets, and let* $R \in \{\dot{\mathrm{R}}, \overline{\mathrm{R}}\}$*. Let* $domain \in \{positive, positive\ and\ negative\}$*.*

An $(\alpha, \mathsf{X}, k, \Sigma, \Gamma, R)$-inference optimization problem from *domain* data *is defined as follows:*

- *The input is a domain R-presentation S of an unknown target restarting automaton over* Γ*, a working alphabet* Γ*, and an input alphabet* Σ*.*
- *The goal of the problem is to find an* X*-automaton* $M = (\Sigma, \Gamma, I)$ *having the size of the window at most k, consistent with S, such that* $\alpha(M)$ *is the smallest among all such automata. For* $k = \infty$*, the size of the window of M is not limited.*

An $(\alpha, \mathsf{X}, k, \Sigma, \Gamma, R)$-inference decision problem from *domain* data *is defined as follows:*

- *The input is a number* $n \in \mathbb{N}$*, a domain R-presentation S of an unknown target restarting automaton over* Γ*, a working alphabet* Γ*, and an input alphabet* Σ*.*
- *The goal of the problem is to decide whether there is an* X*-automaton* $M = (\Sigma, \Gamma, I)$ *having the size of the window at most k, consistent with S, such that* $\alpha(M) \leq n$*. For* $k = \infty$*, the size of the window of M is not limited.*

We will use the function α in this definition as a measure of the size of automata:

- In the case of the decision problem, we are interested only in automata giving a sufficiently low value of α, i.e. we ask whether there is a small automaton.

- In the case of the optimization problem, individual automata are evaluated using α and then we want the automaton giving the lowest possible value of α, i.e. we ask for the smallest automaton.

In Chapter 5 we will study the complexity of this problem. We will sometimes consider only its decision version. This is a common approach when proving NP-hardness of problems. In the case of the decision problem, the question is whether an automaton having the α-value less than a given limit exists. One can choose several different ways to define α. We are particularly interested into the following two as they seem quite natural.

The following measure was presented in [52], where also a measure based on the size of regular expressions is presented.

Definition 4.1.2 ([52])**.** *For a given restarting automaton M, let* $\mathsf{isize}(M)$ *denote the number of meta-instructions of M.*

We can also base such a measure on the size of representations of languages used in meta-instructions of a given restarting automaton.

Definition 4.1.3. *Let* size *be a complexity measure defined as follows (let Γ be an alphabet):*

- *Let $L \subseteq \Gamma^*$ be a regular language. Then* $\mathsf{size}(L)$ *is the number of states of the minimal DFSA accepting L (i.e. the DFSA accepting L and having the minimal number of states among all such automata).*
- *Let $u \in \Gamma^*$ be a word. Then* $\mathsf{size}(u) = \mathsf{size}(\{u\}) = |u| + 1$.
- *Let $i = (L, \mathrm{Accept})$ be an accepting meta-instruction. Then* $\mathsf{size}(i) = \mathsf{size}(L)$.
- *Let $i = (L_\ell, u \to v, L_\mathrm{r})$ be a rewriting meta-instruction. Then* $\mathsf{size}(i) = \mathsf{size}(L_\ell) + \mathsf{size}(u) + \mathsf{size}(v) + \mathsf{size}(L_\mathrm{r})$.
- *Let $M = (\Sigma, \Gamma, I)$ be a restarting automaton. Then* $\mathsf{size}(M) = \Sigma_{i \in I} \mathsf{size}(i)$.

Trying to find an automaton having small isize or size complexity corresponds to the idea of Occam's razor. The size can be considered better from this point of view as, contrary to the isize, it takes account of the complexity of the languages contained in the meta-instructions.

4.2 Definition for Inference Algorithm

The problem definitions presented in Section 4.1 are close to those considered by many researchers. The problem from Definition 4.1.1 will be used in Chapter 5 where we will show the complexity of this general problem. As it turns out, this problem is quite complex. Further, we will consider only a special case of that problem. At first, we will learn from positive samples only. Moreover, the input to the problem will be even more restricted. We will learn from the so-called fully positive analysis by reduction samples.

Definition 4.2.1. *A* fully positive analysis by reduction sample *(FPABR sample) for a language L is a finite sequence $\{w_i\}_{i=0}^{n}$, $w_i \in L$ for all $i \in \{0, \ldots, n\}$, and $|w_i| > |w_{i+1}|$ for all $i \in \{0, \ldots, n-1\}$.*

This also influences our approach to the learning problem. Based on the results from Chapter 5, we will no longer consider the α function (in Chapter 5 we will show that in the case of learning from positive samples only and when considering $\alpha = \mathsf{isize}$ or $\alpha = \mathsf{size}$, the solution of the problems from Definition 4.1.1 can be found in polynomial time — but at the same time, the usefulness of the optimal solution is questionable). Instead of looking for some other measure to be used as α, we will just have the Occam's razor in mind, we will propose an inference method, and then we will evaluate its performance in benchmarks, where the outputs of the learning algorithm will be used to classify unseen testing samples (both positive and negative samples).

Definition 4.2.2. *An* ABR inference from positive samples problem *is defined as follows:*

- *The input consists of a finite set of FPABR samples S, an input alphabet Σ, and a working alphabet Γ.*
- *The goal is to find an* RRWW*-automaton $M = (\Sigma, \Gamma, I)$ consistent with S, i.e. performing all the given reductions and accepting all the given simple words by some of its accepting meta-instructions. I.e. for each FPABR sample $\{w_i\}_{i=0}^{n} \in S$, it holds $w_0 \vdash_M w_1 \vdash_M \cdots \vdash_M w_n$ and $w_n \in E$ for some accepting meta-instruction $(E, \mathrm{Accept}) \in I$.*

Thus, different inference algorithms for solving this problem may return different results despite using the same input samples. The only requirement is that the returned automata are consistent with the input samples. They are not required to be somehow small in any sense.

5. Problem Complexity

The problems which are solvable in (deterministic) polynomial time are considered to be effectively solvable. Therefore, we will analyze the complexity of solving several language learning problems to see if they belong to this class. The analysis of complexity may itself give an algorithm for solving the problem effectively. On the other hand, it may also prove that the given problem is too hard to be solved by current computers (e.g. NP-hard [40]) or even to be solved at all. If so, it is then reasonable to try to find approximate solutions of the problem or to use time-consuming methods if an exact solution is needed. Therefore, we dedicate this chapter to the analysis of complexity of the problem of inferring restarting automata. We particularly analyze the problem from Definition 4.1.1.

A common approach to show that an optimization problem is hard is to prove that the corresponding decision problem is NP-hard. On the other hand, to show that an optimization problem can be solved effectively, we need to consider the original optimization problem. Therefore, we presented both versions of the problem in the definition mentioned above.

In Section 5.1 we introduce some basic tools that will be used to analyze the problem complexity. Section 5.2 presents the actual analysis of problem complexity. The results are briefly discussed in Section 5.3.

5.1 Graph Coloring Problem

Some of the results presented in this chapter show that several problems are NP-hard [40]. A common approach to prove NP-hardness of a problem is to reduce some known NP-complete problem to the problem under examination. Here we use the graph coloring problem introduced below in the place of the known NP-complete problem to be reduced to our problem.

Let us first informally introduce the complexity theory related to NP-hardness (for details see e.g. [36]). In the complexity theory, a decision problem Q is often a language over $\{0,1\}$. The goal is to determine whether a given word belongs to Q or not. The complexity of this task is investigated to see if it can be solved effectively, i.e. in polynomial time. One of the fundamental classes of problems is known as NP. It contains all decision problems that can be solved by a non-deterministic Turing machine in polynomial time. It is sometimes possible to solve one problem easily once we know how to solve another problem. It is said that a decision problem Q' can be reduced to a decision problem Q in polynomial time if there is a function f (computable by a deterministic Turing machine in polynomial time) such that for all $w \in \{0,1\}^*$, it holds $w \in Q'$ if and only if $f(w) \in Q$. The class NP is of particular interest because it contains many practical problems that we are not able to solve effectively nowadays. It is known that some problems are somehow as hard as all of the problems present in the class NP. We say that a decision problem Q is NP-hard if all decision problems $Q' \in \text{NP}$ can be reduced to Q in polynomial time. In addition, if Q is a member of NP, it is called NP-complete. Thus, NP-complete problems can be seen as the hardest among all members of NP. Now, in order to prove that some new problem Q_{new} is NP-hard, it obviously suffices to show that some NP-complete problem

can be reduced to Q_{new} in polynomial time.

Several known NP-complete problems are related to graphs. We introduce one of them now, and then we will use it in the following proofs. The problem consists in coloring vertices of a graph in such a way, that no adjacent vertices (i.e. vertices present on the same edge) are of the same color.

Definition 5.1.1 ([40]). *A* coloring *of a given graph $G = (V, E)$ is a function $c : V \to \mathbb{N}$ such that for each edge $\{u, v\} \in E$, it holds $c(u) \neq c(v)$. The value $c(v)$ is called the* color *of the vertex v.*

Some problems can be solved by coloring a given graph using some limited number of colors. This corresponds to the following problem.

Definition 5.1.2 ([40]). *Let C be a positive integer. The graph coloring problem for given number of colors C is defined as follows:*

- ***Input:*** *a graph G.*
- ***Question:*** *Is there a coloring of G using at most C different colors?*

It is well known that the graph coloring problem belongs to the class of NP-complete problems.

Theorem 5.1.3 ([40]). *The graph coloring problem for $C = 3$ is NP-complete.*

This theorem will be used in proofs presented in Section 5.2 — we will reduce the graph coloring problem to the problems considered there.

5.2 Complexity Results

This section presents complexity results for several versions of the problem of learning restarting automata as defined in Definition 4.1.1. We will show that the problem to find a restarting automaton having the minimal representation is NP-hard. The consequence of this is that it is reasonable to try to use approximation or time-consuming methods to solve those problems.

We will divide our survey according to the considered function α. We start our study with $\alpha = \mathsf{isize}$. We will measure the number of meta-instructions used by a resulting restarting automaton. In the next theorem we show that the problem of finding a restarting automaton consistent with a given presentation and having the minimal number of meta-instructions is hard.

At first, we make a simple note. Let $\{(E_1, \text{Accept}), \ldots, (E_n, \text{Accept})\}$ be the set of all accepting meta-instructions of a given restarting automaton. Then, thanks to the closure properties of regular languages, we can replace all these meta-instructions by a single accepting meta-instruction $(\bigcup_{i=1}^{n} E_i, \text{Accept})$ without changing the language accepted by M or the performed ABR. Hence, at most one accepting meta-instruction is sufficient for any considered type of restarting automata. Thus, an attempt to minimize the number of meta-instructions corresponds to the minimization of the number of rewriting meta-instructions.

On the other hand, let $\{(\{0^i; i \geq 0\}, 01 \to \lambda, \{1^i; i \geq 0\}), (\{2^i; i \geq 0\}, 01 \to \lambda, \{3^i; i \geq 0\})\}$ be the set of rewriting meta-instructions of a given restarting automaton. These meta-instructions cannot be replaced by a single meta-instruction of the form $i = (E_\ell, 01 \to \lambda, E_\text{r})$. Otherwise, from $0 \in E_\ell$ and $3 \in E_\text{r}$

it follows that $(0, 01 \to \lambda, 3) \in i$, however, this reduction can not be performed by any of the original meta-instructions. So we can not replace those meta-instructions this way without changing the ABR performed by this automaton.

Below we will need a mapping from natural numbers and sets of natural numbers to words over $\{0, 1\}$. For this purpose, we use the following simple definition.

Definition 5.2.1. *Let* $\mathrm{b}(n) = 0^n$ *for* $n \in \mathbb{N}$. *Let* $\mathrm{B}(X) = \{\mathrm{b}(i); i \in X\}$ *be the extension of* b *onto the sets of natural numbers.*

Let us start the actual analysis of the problem complexity.

Theorem 5.2.2. *Let* $\mathsf{X} \in \{\mathsf{RRWW}, \mathsf{RRW}, \mathsf{RR}\}, k \in \mathbb{N} \cup \{\infty\}, R \in \{\dot{R}, \overline{R}\}$. *The* $(\mathsf{isize}, \mathsf{X}, k, \{0, 1\}, \{0, 1\}, R)$*-inference decision problem from positive and negative data is NP-hard.*

Proof. As stated previously, the problem whether there is a 3-coloring of a given graph is known to be NP-complete [40]. We will show a polynomial transformation of this problem onto the problem stated in the theorem (any combination of the parameters in the definition of the problem can be used).

Let a graph $G = (V, E)$ where $V = \{1, \ldots, n\}$ for some positive integer n be an instance of the graph coloring problem for $C = 3$ colors (we will use C instead of the constant 3 to avoid possible confusion when this value is used in expressions).

We will associate a pair of vertices (i, j) with a reduction by $\mathrm{r}(i, j) = (0^i, 1 \to \lambda, 0^j)$ in the case of $R = \dot{\mathrm{R}}$ and by $\mathrm{r}(i, j) = (0^i 10^j \Rightarrow 0^i 0^j)$ in the case of $R = \overline{\mathrm{R}}$. Using this function, we define an input for $(\mathsf{isize}, \mathsf{X}, k, \{0, 1\}, \{0, 1\}, R)$-inference problem from positive and negative data as follows:

- $S^+ = S^- = \emptyset$,
- $R^+ = \{\mathrm{r}(i, i); i \in V\}$, and
- $R^- = \{\mathrm{r}(i, j); \{i, j\} \in E \text{ and } i < j\}$.

At first, it may seem quite strange that no words are required to be accepted or rejected by the target automaton (note that $S^+ = S^- = \emptyset$). However, recall the definition of consistence of a restarting automaton with given input samples (Definition 2.2.5). In this proof we decided to focus on whether the restarting automaton performs the given reductions, ignoring the language being actually accepted.

As $i, j \leq n$ (n being the number of vertices), computing $\mathrm{r}(i, j)$ requires no more than $\mathrm{O}(n)$ steps. The input is created in time $\mathrm{O}(n^3)$. The size of the graph coloring input is $\mathrm{O}(n^2)$. Thus, this transformation requires polynomial amount of time.

We will show below that there is a C-coloring of G if and only if there exists an X-automaton M with the size of the window at most k, consistent with S, and such that $\mathsf{isize}(M) \leq C$.

Let $c : V \to \{1, \ldots, C\}$ be a coloring of G, i.e. a solution of the graph coloring problem. Let us create an RR-automaton (and thus also an RRW-automaton and an RRWW-automaton) $M = (\{0, 1\}, \{0, 1\}, I)$, where $I = \{(\mathrm{B}(c^{-1}(z)), 1 \to$

$\lambda, \mathrm{B}(c^{-1}(z))); z = 1, \ldots, C\}$ (here $c^{-1}(z)$ means the set of vertices having the color z).

Let us prove the consistency of this automaton with the input. The automaton M surely accepts all words from S^+ and rejects all words from S^-.

Let $(0^i, 1 \to \lambda, 0^i) \in R^+$ in the case of $R = \dot{\mathrm{R}}$ and $0^i10^i \Rightarrow 0^i0^i \in R^+$ for $R = \overline{\mathrm{R}}$. Let $c(i) = z$, thus $0^i \in \mathrm{B}(c^{-1}(z))$, thus $(0^i, 1 \to \lambda, 0^i) \in (\mathrm{B}(c^{-1}(z)), 1 \to \lambda, \mathrm{B}(c^{-1}(z))) \in I$ for $R = \dot{\mathrm{R}}$ and $(0^i10^i \Rightarrow 0^i0^i) \in (\mathrm{B}(c^{-1}(z)), 1 \to \lambda, \mathrm{B}(c^{-1}(z))) \in I$ for $R = \overline{\mathrm{R}}$. Thus, the requirement related to R^+ is satisfied as well.

Let us suppose there is some located reduction $(0^i, 1 \to \lambda, 0^j)$ in R^- such that I contains an instruction r, where $(0^i, 1 \to \lambda, 0^j) \in r$ for $R = \dot{\mathrm{R}}$. For $R = \overline{\mathrm{R}}$, let us suppose there is some sliding reduction $(0^i10^j \Rightarrow 0^i0^j)$ in R^- such that I contains a meta-instruction r, where $(0^i10^j \Rightarrow 0^i0^j) \in r$. Thus, $c(i) = c(j)$ according to the definition of I. On the other hand, $(0^i, 1 \to \lambda, 0^j) \in R^-$ for $R = \dot{\mathrm{R}}$ and $(0^i10^j \Rightarrow 0^i0^j) \in R^-$ for $R = \overline{\mathrm{R}}$ implies $\{i, j\} \in E$, thus $c(i) \neq c(j)$. This forms a contradiction and thus the requirement for R^- is also satisfied.

In summary, the restarting automaton M is consistent with the input. Obviously, $\mathsf{isize}(M) \leq C$.

Let M be an X-automaton having the size of the window at most k, consistent with S, and having at most C meta-instructions. W.l.o.g. the size of the window is equal to 1 and $\mathsf{X} = \mathsf{RR}$. Meta-instructions rewriting more than one symbol can be either removed (if they do not perform any reduction from R^+) or the rewritten and replacement subwords can be shortened in such a way that the rewriting meta-instruction just replaces 1 with λ. The power of RRW-automata and RRWW-automata is not needed as we have the window of size one and we use no auxiliary symbol in S. We will show that then there is a C-coloring of G.

Let $M = (\Sigma, \Gamma, I)$, $I = \{i_1, \ldots, i_s\}$, for some $s \leq C$. Let $\mathrm{c}(i) = \min\{k; \mathrm{r}(i, i) \in i_k\}$ (the considered set is surely non-empty, as $\mathrm{r}(i, i) \in R^+$ and thus at least one meta-instruction from I must perform the reduction $\mathrm{r}(i, i)$). This forms a coloring as if there is an edge $\{i, j\} \in E$ (where $i < j$) such that $\mathrm{c}(i) = \mathrm{c}(j)$, this would mean that $\mathrm{r}(i, i), \mathrm{r}(j, j) \in i_t$ for some t and thus also $\mathrm{r}(i, j) \in i_t$. This is a contradiction with the consistency of the automaton as $\mathrm{r}(i, j) \in R^-$.

Also, the number of used colors is at most $s \leq C$. Therefore, we have the desired coloring of G.

□

It remains an open question whether the problem considered in the above theorem belongs to NP. It would suffice to show that we are able to guess the right automaton in polynomial time and then check its consistency with input samples in polynomial time. The problem is that if we guess an automaton M, it is not easy to check that it rejects all samples from S^-. Note that M could e.g. reduce a word $w \in S^-$ such that $|w| = 2n$ for some $n > 0$ into 2^n words — imagine that M has the ability to replace any subword 00 with either 0 or 1. Then the word 0^{2n} can be easily reduced by M into any word from $\{0, 1\}^n$. Thus, the straightforward approach of performing all possible computations on w is obviously not polynomial!

The previous theorem states that the problem of deciding whether there is a restarting automaton consistent with a given presentation and having the number of meta-instructions below the given limit is NP-hard.

Another natural way to measure the complexity of an automaton is the size of its representation as defined by the function size. Below we analyze the complexity of inferring restarting automata while trying to achieve the size complexity below a given limit.

Theorem 5.2.3. *Let* $\mathsf{X} \in \{\mathsf{RR}, \mathsf{RRW}, \mathsf{RRWW}\}, k \in \mathrm{N} \cup \{\infty\}, R \in \{\dot{\mathrm{R}}, \overline{\mathrm{R}}\}$. *The* $(\mathsf{size}, \mathsf{X}, k, \{0,1\}, \{0,1\}, R)$*-inference decision problem from positive and negative data is NP-hard.*

Proof. Similarly as above, we will transform the graph coloring problem onto the given problem.

Let a graph $G = (V, E)$ where $V = \{1, \ldots, n\}$ be an instance of the graph coloring problem for $C = 3$ colors (we will use C instead of the constant 3 to avoid possible confusion when this value is used in expressions).

The idea is to force the automata learning algorithm to minimize the number of meta-instructions. To do this, we prepend the positive reductions from the previous proof by *a bubble*, i.e. by a prefix that forces individual meta-instructions to have big size-complexity. This way, minimizing the number of meta-instructions (isize) is more proficient (in the sense of minimizing the size complexity) than minimizing the meta-instructions alone.

Let $\mathrm{r}(i, j, \beta) = (1^\beta 0^i, 1 \to \lambda, 0^j)$ for $R = \dot{\mathrm{R}}$ and let $\mathrm{r}(i, j, \beta) = (1^\beta 0^i 1 0^j \Rightarrow 1^\beta 0^i 0^j)$ for $R = \overline{\mathrm{R}}$, where $\beta \in \mathbb{N}$. The prefix 1^β corresponds to the above mentioned bubble.

Define a presentation S for $(\mathsf{size}, \mathsf{X}, k, \{0,1\}, \{0,1\}, R)$-inference problem from positive and negative data as follows:

- $S^+ = S^- = \emptyset$,
- $R^+ = \{\mathrm{r}(i, i, z); i \in V\}$, and
- $R^- = \{\mathrm{r}(i, j, z); \{i, j\} \in E \text{ and } i < j\} \cup \bigcup_{i \in V} \bigcup_{m \in \{0, \ldots, z-1\}} \mathrm{r}(i, i, m)$,

where $z = C(n \cdot (\mathsf{size}(0^n) + \mathsf{size}(0^n)) + 4) + 1$ (you will see the importance of this value later).

Note that R^- is composed of two parts. The first one corresponds to that one already used in the previous proof and it corresponds to the edges of G. Its purpose is the same as in the previous proof, but in addition it contains the bubbles. The second part is our way to force meta-instructions to have big size-complexity, as will be shown below.

The value of z is chosen to fit into an expression used later. Another important value is defined as $\ell = z + (n(\mathsf{size}(0^n) + \mathsf{size}(0^n)) + 4)$. We will show below that there is a C-coloring of G if and only if there is an X-automaton having the size of the window at most k, consistent with S, and having the value of size at most $C \cdot \ell$.

The value of $\mathrm{r}(i, j, \beta)$ can be computed in time $O(n^2)$ for all combinations of arguments values considered in this proof. To transform the instance of the graph coloring problem into the instance of $(\mathsf{size}, \mathsf{X}, k, \{0,1\}, \{0,1\}, R)$-inference problem takes no more than $O(n^5)$ steps.

The size of the graph coloring input is $\mathrm{O}(n^2)$. Thus, the transformation runs in polynomial time with respect to the size of the original input.

At first, let $\mathrm{c} : V \to \{1, \ldots, C\}$ be a coloring of G. As in the proof of the previous theorem, we can transform this coloring into a corresponding restarting automaton M such that $\mathsf{isize}(M) \leq C$: Let $M = (\{0,1\}, \{0,1\}, I)$, where $I = \{(1^z \cdot \mathrm{B}(\mathrm{c}^{-1}(m)), 1 \to \lambda, \mathrm{B}(\mathrm{c}^{-1}(m))); m = 1, \ldots, C\}$.

We will prove the consistency of M with S in the case of the located reductions at first.

The restarting automaton M surely accepts all words from S^+ and rejects all words from S^- as $S^+ = S^- = \emptyset$.

Let $(1^z 0^i, 1 \to \lambda, 0^i) \in R^+$. Let $\mathrm{c}(i) = c_0$, thus $0^i \in \mathrm{B}(\mathrm{c}^{-1}(c_0))$, thus $(1^z 0^i, 1 \to \lambda, 0^i) \in (1^z \cdot \mathrm{B}(\mathrm{c}^{-1}(c_0)), 1 \to \lambda, \mathrm{B}(\mathrm{c}^{-1}(c_0))) \in I$. Thus, the requirement related to R^+ is satisfied as well.

Let $r \in R^-$. There are two possible cases:

1. Let $r = (1^z 0^i, 1 \to \lambda, 0^j)$. Thus, $\{i, j\} \in E$ and $\mathrm{c}(i) \neq \mathrm{c}(j)$. If M performs r, then $\mathrm{c}(i) = \mathrm{c}(j)$ according to the definition of I. This forms a contradiction.

2. Let $r = (1^{z'} 0^i, 1 \to \lambda, 0^i)$ for some $z' < z$. Let us suppose r is performed by a meta-instruction $(L_\ell, 1 \to \lambda, L_\mathrm{r})$ of I. Obviously, $1^{z'} 0^i \in L_\ell$. By inspecting the definition of I, we obtain that every member of L_ℓ starts with 1^z, but the longest prefix consisting of 1's of the considered word has less than z symbols and thus is not a member of L_ℓ. Thus, no reduction of this type is performed.

In summary, the automaton M is consistent with the input. The case of sliding reductions is analogous.

It remains to estimate the size-complexity of M. Let p be a meta-instruction of M. Then $\mathsf{size}(p) \leq z + 1 + n \cdot \mathsf{size}(0^n) + 2 + 1 + n \cdot \mathsf{size}(0^n) = \ell$, as the minimal DFSA accepting the bubble is of size at most $z + 1$, the minimal DFSA accepting the left context (except the bubble) of the meta-instructions p has at most $n \cdot \mathsf{size}(0^n)$ states, the minimal DFSA accepting only the word 1 has 2 states, the minimal DFSA accepting only the empty word has 1 state, and finally, the minimal DFSA accepting the right context of the meta-instruction p has at most $n \cdot \mathsf{size}(0^n)$ states. Thus, we have an X-automaton having the size of the window at most k, consistent with S, and having $\mathsf{size}(M) \leq C \cdot \ell$.

On the other hand, let M be an X-automaton having the size of the window at most k, consistent with S, and such that $\mathsf{size}(M) \leq C \cdot \ell$. W.l.o.g. it holds $X = \mathsf{RR}$. In addition, consider M having the smallest number of meta-instructions among such automata. We will show that $\mathsf{isize}(M) \leq C$.

Let $t = (L_\ell, u1v \to uv, L_\mathrm{r})$ (where $u, v \in \{0,1\}^*$) be any meta-instruction of M (note that in the case of sliding reductions, we can not assume $u = v = \lambda$). Obviously, this meta-instruction performs at least one of the reductions from R^+, let it be (we express it here using the sliding reduction notation) $1^z 0^i 1 0^i \Rightarrow 1^z 0^i 0^i$. Let us express it as a located reduction using $(w_\ell, u1v \to uv, w_\mathrm{r})$ for some $w_\ell \in L_\ell$ and $w_\mathrm{r} \in L_\mathrm{r}$. Obviously, $\mathsf{size}(t) \geq \mathsf{size}(L_\ell)$. We will prove that $\mathsf{size}(t) \geq z$.

Note that we can not use the technique of the previous proof where prefix and suffix were shifted to the left and right contexts, respectively. Otherwise, we would possibly increase the size of t and that is unappropriate for the purpose of this proof.

Let m be the length of the longest prefix of w_ℓ consisting of 1's only. We analyze two cases:

- If $m < z$, then obviously $w_\ell = 1^m \in L_\ell$. We will show that $\mathsf{size}(L_\ell) \geq m$. Let us suppose the opposite holds, i.e. there is a DFSA M_{L_ℓ} accepting L_ℓ having less than m states. Then $(\exists r, s \in \{0, \ldots, m-1\})((r < s) \& (1^r \sim 1^s))$, where $\sim$ is the equivalence given by the Nerode's theorem (Theorem 1.1.26). Thus, we further have $1^{r+m-s} \sim 1^{s+m-s} = 1^m \in L_\ell$ and thus $1^{r+m-s} \in L_\ell$. However, $r + m - s = r - s + m < m$. Therefore, even the reduction $(1^{r-s+m}, u1v \to uv, w_\mathrm{r})$ for $R = \dot{\mathrm{R}}$ and $(1^{r-s+m}u1vw_\mathrm{r} \Rightarrow 1^{r-s+m}uvw_\mathrm{r})$ for $R = \overline{\mathrm{R}}$ will be performed as well as the original ones $(1^m, u1v \to uv, w_\mathrm{r})$ and $(1^m u1vw_\mathrm{r} \Rightarrow 1^m uvw_\mathrm{r})$. However, $1^{r-s+m}u1vw_\mathrm{r}$ starts with less than z 1's and thus the meta-instruction t performs a negative reduction. That is a contradiction. Therefore, $\mathsf{size}(L_\ell) \geq m$. As the remaining 1's of the bubble necessarily form a prefix of u, we have $\mathsf{size}(t) \geq z$.

- In the case of $m \geq z$, we will prove that $\mathsf{size}(L_\ell) \geq z$. Let us suppose the opposite holds, i.e. there is a DFSA M_{L_ℓ} accepting L_ℓ having less than z states. Then $(\exists r, s \in \{0, \ldots, z-1\})((r < s) \& (1^r \sim 1^s))$, where $\sim$ is the equivalence given by Nerode's theorem (Theorem 1.1.26). Thus, we further have $1^{r+z-s} \sim 1^{s+z-s} = 1^z \in L_\ell$ and thus $1^{r+z-s} \in L_\ell$. However, $r+z-s = r-s+z < z$. Therefore, even the reduction $(1^{r-s+z}u', u1v \to uv, w_\mathrm{r})$ for $R = \dot{\mathrm{R}}$ and $(1^{r-s+z}u'u1vw_\mathrm{r} \Rightarrow 1^{r-s+z}u'uvw_\mathrm{r})$ for $R = \overline{\mathrm{R}}$ will be performed as well as the original ones $(1^z u', u1v \to uv, w_\mathrm{r})$ and $(1^z u'u1vw_\mathrm{r} \Rightarrow 1^z u'uvw_\mathrm{r})$ $(u' \in \{0,1\}^*)$. However, $1^{r-s+z}u'u1vw_\mathrm{r}$ starts with less than z 1's and thus the meta-instruction t performs a negative reduction. That is a contradiction. Therefore, $\mathsf{size}(L_\ell) \geq z$ and finally $\mathsf{size}(t) \geq z$.

Thus, in both cases it holds $\mathsf{size}(t) \geq z$. It further holds $\mathsf{size}(M) \geq \mathsf{isize}(M) \cdot z$.

Let us suppose $\mathsf{isize}(M) > C$, i.e. $\mathsf{isize}(M) \geq C+1$. Then $\mathsf{size}(M) \geq (C+1) \cdot z$. Let us compute: $z = C(n \cdot (\mathsf{size}(0^n) + \mathsf{size}(0^n)) + 4) + 1 > C(n \cdot (\mathsf{size}(0^n) + \mathsf{size}(0^n)) + 4)$, thus $C \cdot z + z > C \cdot z + C(n \cdot (\mathsf{size}(0^n) + \mathsf{size}(0^n)) + 4)$, thus $(C+1)z > C \cdot \ell$. In summary, $\mathsf{size}(M) > C \cdot \ell$. This forms a contradiction and thus $\mathsf{isize}(M) \leq C$.

Now, analogically to the previous proof, we will transform this automaton M into a C coloring. At first, we transform rewriting meta-instructions rewriting more than one symbol into rewriting meta-instructions just deleting the symbol 1. This does not change the number of meta-instructions. Also, if u could be directly reduced to v before the transformation, it is still possible to directly reduce u to v after the transformation, and vice versa.

Let M have the set of meta-instructions $I = \{i_1, \ldots, i_r\}$. Let

$$\mathrm{c}(i) = \min\{m; \mathrm{r}(i, i, z) \in i_m\}.$$

Obviously, this forms a coloring as if there is an edge $\{i, j\} \in E, i < j$ such that $\mathrm{c}(i) = \mathrm{c}(j)$, this would mean that $\mathrm{r}(i, i, z), \mathrm{r}(j, j, z) \in i_m$ for some m and thus also $\mathrm{r}(i, j, z) \in i_m$. This is in a contradiction with consistency of the automaton as $\mathrm{r}(i, j, z) \in R^-$. Also, the number of used colors is at most $r \leq C$.

□

Again, the same reason as in the case of Theorem 5.2.2 causes problems in proving the membership of the above analyzed problem in NP.

5.2.1 Positive Samples Only

This subsection discusses the problem of inference from positive data only. That means there is neither information about words outside of the target language nor what reductions are considered wrong.

Again, we are interested in learning restarting automata having small size. In the case of both isize and size measures and located reductions, the problem can be proven to be of polynomial complexity.

Theorem 5.2.4. *Let $\alpha \in \{\mathsf{isize}, \mathsf{size}\}$ be a complexity measure, let $X \in \{\mathsf{RRWW}, \mathsf{RRW}, \mathsf{RR}\}$, $k \in \mathbb{N} \cup \{\infty\}$, and let $\Sigma \subseteq \Gamma$ be given alphabets. The $(\alpha, X, k, \Sigma, \Gamma, \dot{\mathrm{R}})$-inference optimization problem from positive data can be solved in polynomial time.*

Proof. It is quite easy to see that the problem can be solved in polynomial time. Let us have an $\dot{\mathrm{R}}$-presentation $(S^+, \emptyset, R^+, \emptyset)$ of a restarting automaton over Γ. In the case of $\alpha = \mathsf{isize}$, it suffices to group supplied reductions from R^+ according to the rewrite $u \to v$ they perform and count the number of such groups. The resulting number is the lower bound for the number of rewriting meta-instructions needed to describe restarting automaton consistent with given presentation. For each such located reduction containing rewrite $u \to v$, we create the rewriting meta-instruction $(\Gamma^*, u \to v, \Gamma^*)$. If needed, we add an accepting meta-instruction. No more meta-instructions are needed. Also note that to check that the supplied value k is sufficient to allow all required reductions to be performed, we need polynomial amount of time only.

The case of $\alpha = \mathsf{size}$ is similar. Obviously, we need at least the same number of meta-instructions as before. The number of meta-instructions is surely sufficient as shown above. In addition, each context in the meta-instructions of the solution for the case of $\alpha = \mathsf{isize}$ can be represented using a one-state DFSA. It is the case also for the language contained in the accepting meta-instruction, if there is any. We know that at least the current number of meta-instructions is surely needed and also that all the rewrites $u \to v$ have to be present in some meta-instruction. So the overall size complexity of any restarting automaton consistent with the input cannot be lower than the one of our solution for the case $\alpha = \mathsf{isize}$.

□

So we could conclude that the problem of inferring from positive data is much easier than inferring from both positive and negative data. But naturally, in practice, our intent is to learn the target language or the target ABR. The key of using Occam's razor is just a hint. When asked if it is easier to learn something from both positive and negative data or from data where negative samples are missing, it is quite clear that using less data could hardly improve our performance. From this point of view, the problem of inferring from positive data only is much more complex.

The case of positive presentations containing sliding reductions is considerably more complex. We will analyze the complexity of minimizing the number of meta-instructions again.

Let us start with some theorems. The first one states that if two located reductions r_1 and r_2 of some particular word w lead to the same resulting word w' and r_1 replaces a shorter subword than r_2, then it is possible to change r_2

to r_2' by shortening the replaced and replacement words (by removing common prefixes and suffixes) in such a way that r_2' reduces w into w' as well.

Theorem 5.2.5. *Let Γ be an alphabet and let $u, v, v', w, x, y, y', z \in \Gamma^*$ such that*

1. $uvw = xyz$,
2. $uv'w = xy'z$, *and*
3. $|v| < |y|$.

Then there are $\tilde{y}, \tilde{y}', r, s \in \Gamma^*$ such that

(i) $y = r\tilde{y}s$,

(ii) $y' = r\tilde{y}'s$, *and*

(iii) $|\tilde{y}| = |v|$.

Proof. Let $\tilde{y}, \tilde{y}', r, s \in \Gamma^*$ be some words satisfying the conditions (i) and (ii) and such that $\tilde{y}$ is of the minimal length among all such words having the length at least $|v|$. Let us suppose $|\tilde{y}| > |v|$. It follows that the first and the last symbol of $\tilde{y}$ (let us denote them a and b, respectively) differ from the first and the last symbol of $\tilde{y}'$ (let us denote them c and d, respectively), i.e. $a \neq c$ and $b \neq d$. Now we will discuss all three possible cases:

- v contains the particular occurrence of a: As $|v| < |\tilde{y}|$, the rewrite using v' does not affect the suffix bsz. Hence, $d = b$ which is a contradiction.
- v contains the particular occurrence of b: As $|v| < |\tilde{y}|$, the rewrite using v' does not affect the prefix xra. Hence, $a = c$ which is a contradiction.
- v contains neither the particular occurrence of a nor the particular occurrence of b: Either v is contained in $\tilde{y}$ and thus neither the prefix xra nor the suffix bsz is changed, or v is left to a or right to b and thus the suffix bsz or the prefix xra is not changed, respectively. In both cases, we get a contradiction.

Finally we have that $|\tilde{y}| = |v|$ and thus all the conditions are met.

□

Let us look at the just proven theorem. What do the conditions 1 to 3 actually mean? Let $(u, v \to v', w)$ and $(x, y \to y', z)$ be two located reductions such that $u, v, v', w, x, y, y', z \in \Gamma^*$ satisfy all conditions of Theorem 5.2.5. According to the first two conditions, both reductions reduce the same word and the results of both reductions are also the same words. At last, the third condition states that the first reduction replaces a shorter subword than the second one. Under those conditions, the theorem yields that y and y' can be shortened by removing a common prefix r and/or a common suffix s to finally obtain another reduction $(xr, \tilde{y} \to \tilde{y}', sz)$ such that $|v| = |\tilde{y}|$ and it again reduces the original word into the same word as any of the original two reductions.

The following theorem considers the case when there are two located reductions r_1, r_2 reducing w to w'. It claims that it is possible to create a located

reduction r_2' such that it reduces w to w' as well, and in addition, the rewritten subword is located at any chosen place between the places of subwords rewritten by r_1 and r_2, and the length of the word being rewritten is the same in r_2 and r_2'.

Theorem 5.2.6. *Let Γ be an alphabet and let $u, v, v', w, x, y, y', z \in \Gamma^*$ such that*

1. $uvw = xyz$,
2. $uv'w = xy'z$,
3. $|v| = |y|$,
4. $|u| < |x|$, *and*
5. $|v| > |v'|$.

Then for all n such that $|u| < n < |x|$, there exist $r, s, s', t \in \Gamma^*$ satisfying

(i) $|r| = n$,

(ii) $rst = uvw$,

(iii) $rs't = uv'w$, *and*

(iv) $|s| = |v|$.

Proof. W.l.o.g. we take $n = |u| + 1$. We will divide the initial setting according to whether v and y are separated by at least one symbol.

At first, let v and y be separated by a nonempty word q as depicted in Fig. 5.1 (note that the case of $q = \lambda$ is analyzed later in this proof). In this case, it holds $uv'qyz = uvqy'z$ for some $q \in \Gamma^+$. Let $q = ap$ for some $a \in \Gamma, p \in \Gamma^*$. Then let $r = uv_0, s = v_{\overline{0}}a$, and $t = pyz$. Recall that $v_{\overline{0}}$ is the word v without its first symbol. It holds $rst = uv_0 \cdot v_{\overline{0}}a \cdot pyz = uvapyz = uvqyz = uvw$. Consider the following two cases:

- In the case of $v' \neq \lambda$, it holds $v_0 = v_0'$ (because $uv'qyz = uvqy'z$ and $v \neq \lambda$). Then let $s' = v_{\overline{0}}'a$. Then $rs't = uv_0 \cdot v_{\overline{0}}'a \cdot pyz = uv_0' \cdot v_{\overline{0}}'a \cdot pyz = uv'apyz = uv'qyz = uv'w$.
- In the case of $v' = \lambda$, let $s' = \lambda$. Then $rs't = uv_0 \cdot \lambda \cdot pyz = ua \cdot pyz = uqyz = uw = uv'w$ as $v_0 = a$.

At second, let v and y be not separated (as depicted in Fig. 5.2, where $v = \tilde{v}q$ and $y = q\tilde{y}$ for some $q \in \Gamma^*, \tilde{v}, \tilde{y} \in \Gamma^+$). In this case it holds $uv'\tilde{y}z = u\tilde{v}y'z$. Let $r = uv_0, s = v_{\overline{0}}\tilde{y}_0$, and $t = \tilde{y}_{\overline{0}}z$. Then $rst = uv_0 \cdot v_{\overline{0}}\tilde{y}_0 \cdot \tilde{y}_{\overline{0}}z = uv\tilde{y}z = uvw$. We consider two cases:

- In the case of $v' \neq \lambda$, it holds $v_0 = v_0' = \tilde{v}_0$. Let $s' = v_{\overline{0}}'\tilde{y}_0$. Then $rs't = uv_0 \cdot v_{\overline{0}}'\tilde{y}_0 \cdot \tilde{y}_{\overline{0}}z = uv'\tilde{y}z = uv'w$.
- In the case of $v' = \lambda$, it holds $v_0 = \tilde{v}_0 = \tilde{y}_0$. Let $s' = \lambda$. Then it holds $rs't = uv_0 \cdot \lambda \cdot \tilde{y}_{\overline{0}}z = u\tilde{y}z = uw = uv'w$.

□

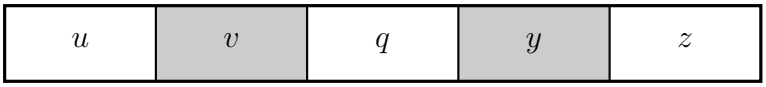


Figure 5.1: Subwords v and y separated by at least one symbol.

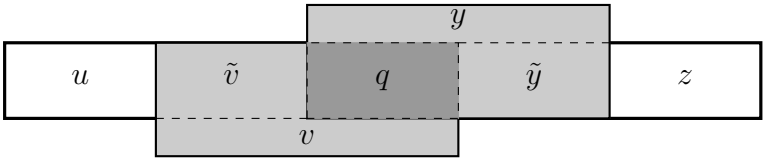


Figure 5.2: Subwords v and y not separated by at least one symbol.

Let us now look at the previous theorem. What do the conditions 1 to 5 actually mean? Let $(u, v \to v', w)$ and $(x, y \to y', z)$ be two reductions such that $u, v, v', w, x, y, y', z \in \Gamma^*$ satisfy all those conditions of Theorem 5.2.6. The first condition states that they both reduce the same original word. The second one states that after the rewrite is performed, the resulting words are the same for both of the reductions. The third condition states that the length of the replaced subword is the same in both given reductions, i.e. $|v| = |y|$. The fourth condition states that the subword replaced by the first reduction occurs earlier in the original word than the subword rewritten by the second reduction. Finally, the fifth condition states that a shorter word is used to replace the original subword. Under these conditions, the theorem states that for any position after v_0 and at the same time before y_0, it is possible to give another reduction that transforms the original word to the same word as obtained by any of the two reductions, but the rewritten subword is located at the chosen position, i.e. the rewriting occurs between the rewritings performed by the original reductions.

Let us now analyze how an ambiguity looks like. Let us consider the sliding reduction $uv_0v_{\overline{0}}aw \Rightarrow uv'aw$ for $u, v, v', w \in \Gamma^*$ and $a \in \Gamma$ (where Γ is an alphabet). We expressed the words in this expression in such a way that suggests one particular located reduction. But note that a particular way of dividing words in such expressions does not relate to how the reduction is actually performed. One way to express the sliding reduction as a located one is $(u, v \to v', aw)$. We may ask when it is possible to express the same sliding reduction as the located reduction $(uv_0, v_{\overline{0}}a \to v'', w)$ for some $v'' \in \Gamma^*$, i.e. shifted by one symbol to the right — we will call this *deferring a rewriting*.

Let us see a particular example, the sliding reduction $00000 \Rightarrow 0000$. One located reduction can be given considering $u = 0, v = 0, v' = \lambda, a = 0,$ and $w = 00$. This yields the located reduction $(0, 0 \to \lambda, 000)$. Thus, the second occurrence of 0 is removed. This rewriting can be deferred by considering $v'' = \lambda$ and taking the located reduction $(00, 0 \to \lambda, 00)$.

Note that things need not be that simple as rewriting to the empty word. Let us consider the sliding reduction $0101 \Rightarrow 001$. It can be divided into $u = \lambda, v = 01, v' = 0, a = 0,$ and $w = 1$. This yields the located reduction $(\lambda, 01 \to 0, 01)$. The other way to perform the sliding reduction $0101 \Rightarrow 001$ can be the located reduction $(0, 10 \to 0, 1)$ where we consider $v'' = 0$.

The following theorem shows when a sliding reduction $uvaw \Rightarrow uv'aw$ $(u, v, v',$

$w \in \Gamma^*, a \in \Gamma$) can be expressed by two different located reductions $(u, v \to v', aw)$ and $(uv_0, v_{\overline{0}}a \to v'', w)$ for some $v'' \in \Gamma^*$. It allows to decide if we can defer the place of a rewriting by one symbol to the right.

Theorem 5.2.7. *Let Γ be an alphabet and let $u, v', w \in \Gamma^*, v \in \Gamma^+, a \in \Gamma$. The following conditions are equivalent:*

- *there is $v'' \in \Gamma^*$ such that $uv_0v''w = uv'aw$*
- *either $(v' = \lambda$ and $a = v_0)$ or $(v' \neq \lambda$ and $v'_0 = v_0)$*

Proof. Let there be $v'' \in \Gamma^*$ such that $uv_0v''w = uv'aw$. Let us consider $v' = \lambda$ first. It follows that $uv_0v''w = uv'aw = uaw$ and finally $v_0 = a$. Or let us consider $v' \neq \lambda$. It follows that $uv_0v''w = uv'aw = uv'_0v'_{\overline{0}}aw$ and thus $v_0 = v'_0$.

On the other hand, let us first consider $v' = \lambda$ and $a = v_0$. Let $v'' = \lambda$. Then $uv_0v''w = uv_0w = uaw = uv'aw$. Or let us consider $v' \neq \lambda$ and $v'_0 = v_0$. Let $v'' = v'_{\overline{0}}a$. Then $uv_0v''w = uv_0v'_{\overline{0}}aw = uv'_0v'_{\overline{0}}aw = uv'aw$.

□

Now let us return to the actual study of the complexity of learning from positive data only. In the case of sliding reductions, we have some freedom in selecting the actual rewriting performed by a reduction. One should take care not to pick a sub-optimal solution. For example, let us consider the following positive $\overline{\mathrm{R}}$-presentation of a restarting automaton over $\Gamma = \{0, 1, 2\}$:

- $S^+ = \emptyset$ (the positive sample words are not needed in this example),
- $S^- = \emptyset$ as we have a positive presentation,
- $R^+ = \{0120 \Rightarrow 0, 1201 \Rightarrow 1\}$, and
- $R^- = \emptyset$ again as we have a positive presentation.

Note that the reduction $0120 \Rightarrow 0$ can be obtained by removing either the subword 012 or the subword 120. Similarly, the latter reduction can be obtained by removing either the subword 120 or the subword 201. Figure 5.3 shows this situation. The upper part contains some rewrites using the empty replacement word. The bottom contains the samples from R^+. An edge means that it is possible to perform a reduction from R^+ by the particular rewrite. Now we can take the rewrites $012 \to \lambda$ and $201 \to \lambda$ to form our solution, i.e. the restarting automaton having rewriting meta-instructions of form $(\Gamma^*, 012 \to \lambda, \Gamma^*)$ and $(\Gamma^*, 201 \to \lambda, \Gamma^*)$. But note that there is a simpler solution using only the rewriting meta-instruction $(\Gamma^*, 120 \to \lambda, \Gamma^*)$.

Below we give an algorithm for finding the optimal solution. The proof of this theorem contains several facts that could be themselves very interesting.

Theorem 5.2.8. *Let $\alpha \in \{\mathsf{isize}, \mathsf{size}\}, X \in \{\mathsf{RRWW}, \mathsf{RRW}, \mathsf{RR}\}, k \in \mathbb{N} \cup \{\infty\}$ and let Σ and Γ be two alphabets. The $(\alpha, X, k, \Sigma, \Gamma, \overline{\mathrm{R}})$-inference optimization problem from positive data can be solved in polynomial time.*

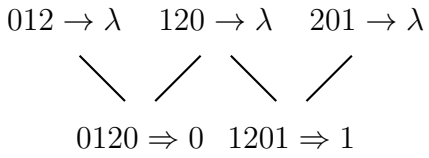


Figure 5.3: A diagram of possible ways to perform reductions by rewrites using the empty replacement word.

Proof. We will start with $\alpha = \mathsf{isize}$. Let S^+ be a set of sample words and let R^+ be a set of sample sliding reductions. We will build a restarting automaton required by the theorem.

If $S^+ = \emptyset$, there will be no accepting meta-instruction, otherwise there will be exactly one, for example $(\Gamma^*, \mathrm{Accept})$. Surely, this choice determines that the automaton will accept either no or all words over Γ regardless of its rewriting meta-instructions. However, in practice, the choice of accepting meta-instructions could be different and there the following steps can be of interest. It remains to specify the rewriting meta-instructions. If $R^+ = \emptyset$, there will be no rewriting meta-instruction. If there is a reduction that can not be performed using the limited length of the window, the problem has no solution. Otherwise, we proceed as follows.

W.l.o.g. we consider only rewriting meta-instructions $(L_\ell, u \to v, L_\mathrm{r})$ such that u and v share neither a common non-empty prefix nor a common non-empty suffix. Note that when v is non-empty, then if w can be reduced to w' using this rewriting meta-instruction, then there the shortest rewriting of w into w' is unique. Reductions requiring such rewriting meta-instructions, i.e. rewriting meta-instructions with non-empty replacement word, will be handled by rewriting meta-instructions $(\Gamma^*, u \to v, \Gamma^*)$. The remaining part of this proof handles reductions that can be performed by rewriting to the empty replacement word, because there is a possibility of ambiguous rewritings.

At first, we build a graph as described above and presented in Fig. 5.3. It is a bipartite graph $G = (U \cup V, E)$ composed of parts U and V. The set U corresponds to members of R^+ such that the reductions can be performed by just removing a subword. Members of V represent members of $\{(w \to \lambda); w \in \Gamma^{\leq k}\}$ (but only a very limited number of such vertices will be actually present in the graph). There is an edge $\{(u \Rightarrow v), (w \to \lambda)\}$ for $(u \Rightarrow v) \in R^+$ and $w \in \Gamma^*$ if there are $x, y \in \Gamma^*$ such that $u = xwy$ and $v = xy$ (i.e. v can be obtained from u by removing the subword w). Now the goal is to find the smallest subset $V_\mathrm{opt} \subseteq V$ such that for each vertex from U, there is a adjacent vertex from V_opt in G. Then for each vertex $(w \to \lambda) \in V_\mathrm{opt}$, we create a rewriting meta-instruction $(\Gamma^*, w \to \lambda, \Gamma^*)$ and include it in the set of meta-instructions of the resulting automaton.

We know that there can be vertices in U having a degree higher than one. Let us consider that case. Let $(u \Rightarrow v)$ be such a vertex. Let $(x, w \to \lambda, ay)$ (where $x, w, y \in \Gamma^*$ and $a \in \Gamma$) be the leftmost (i.e. $|x|$ being the smallest possible) located reduction that transforms $u = xway$ to $v = xay$. As the degree of $(u \Rightarrow v)$ is greater than one, there is another place where a rewriting can occur that performs the reduction $(u \Rightarrow v)$. From Theorem 5.2.6, it follows that $(xw_0, w_{\overline{0}}a \to \lambda, y)$

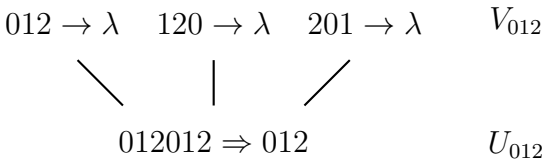


Figure 5.4: An r-ordering of vertices from V_{012}.

also transforms u to v. Then from Theorem 5.2.7, we know that $w_0 = a$. In other words, the word w is a rotation of the word $w_{\overline{0}}a$. If there are other places where a reduction can be performed, then using this idea, we obtain following rotations of w. Thus, a vertex from U is adjacent only to words such that one is a rotation of the other (but not necessarily to all such words).

This allows us to divide the problem to subproblems — we will consider only the subpart $V_w = \{(w' \to \lambda); w' \in \mathrm{rot}(w)\} \subseteq V$ corresponding to rotations of a particular word w. Let U_w denote the vertices adjacent to vertices from V_w. Now the goal is to find the smallest subset $V_{w,\mathrm{opt}}$ of V_w such that each vertex from U_w is adjacent to at least one vertex from $V_{w,\mathrm{opt}}$. Then for each pair $(w' \to \lambda) \in V_{w,\mathrm{opt}}$, we build the rewriting meta-instruction $(\Gamma^*, w' \to \lambda, \Gamma^*)$. Thus, all reductions from U_w will be performed by such meta-instructions. All those meta-instructions will be included in the resulting restarting automaton forming the solution of the whole problem — then surely all reductions from U will be performed by some meta-instruction.

This subproblem is very similar to the set cover problem known to be NP-complete [40]. Fortunately, we deal only with a subproblem of the set cover problem that can be solved in polynomial time, as proven below. We will exploit special relations between vertices present in our problem setting. We were not able to find a published solution for this problem. Therefore, we propose our own method here.

To better understand the proof of the correctness of the algorithm presented below, we will later consider a particular ordering of vertices from V_w. The vertices will be ordered in a row in such a way that neighbouring vertices correspond to words where one is a rotation of the other by exactly one symbol, i.e. if $(r \to \lambda)$ is followed by $(s \to \lambda)$, then $s = r \lll 1$. We call such an ordering an *r-ordering*. This is illustrated in Fig. 5.4 where the word $w = 012$ is considered.

It is important to note how edges leaving a vertex $(u \Rightarrow v) \in U_w$ can look like. We know that if the position where a rewriting occurs is ambiguous, then all locations between the leftmost and the rightmost possible place are possible places of a rewriting as well (from Theorem 5.2.6). It follows that there are no gaps in the row of corresponding rotations of the word being rewritten in the leftmost case. By this we mean that we can rotate the word by one symbol until we obtain the word rewritten in the rightmost position and each intermediate word can be used as a rewritten word in a rewriting performing $(u \Rightarrow v)$.

This idea is shown in Fig. 5.5 in the case of $01201 \Rightarrow 01$ — all vertices from V_{012} are adjacent to this vertex. You may be confused by the situation with the vertex labeled $2012 \Rightarrow 2$. It is adjacent to the vertex labelled $012 \to \lambda$, then the vertex labelled $120 \to \lambda$ is skipped, and the vertex labelled $201 \to \lambda$ is again adjacent. How is that possible? Actually, there is no gap in the sense described

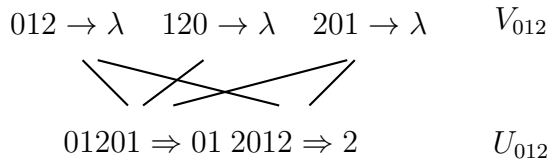


Figure 5.5: Possible configurations of edges leaving the same vertex from U_{012}.

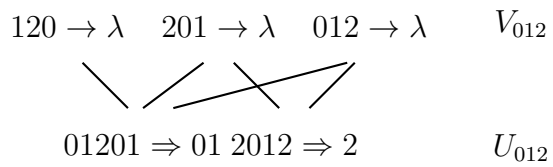


Figure 5.6: An alternative r-ordering of V_{012} leading to no false gaps.

above (we will therefore call the present state *a false gap*), both the word 201 and its rotation 012 are adjacent to this vertex. However, such false gaps would cause problems in our proposed algorithm. Therefore, we will show how to avoid them.

One idea is to choose a different ordering of V_w — in the previous case it suffices to start with 120 as depicted in Fig. 5.6. However, there are cases where no such ordering is possible. It is depicted in Fig. 5.7.

Therefore, we propose the following solution. We try all possible r-orderings (note there are at most $|V_w|$ of them). For each r-ordering, we consider the smallest vertex $v_0 \in V_w$ as a member of the solution. Surely, at least one of the vertices of V_w is present in the solution. We can then safely remove v_0 and all adjacent vertices (from U_w) from the present graph and solve the remaining problem, this time containing no false gap. This is because vertices causing a false gap in the original graph were surely adjacent to v_0 as we already know there is no gap. Let the set of vertices of the obtained graph consist of parts $U'_w \subseteq U_w$ and $V'_w \subset V_w$. Eventually, we select the best solution among the solutions obtained for all considered r-orderings of V_w.

In what follows, we consider a graph containing no false gaps. The r-ordering of remaining vertices from V'_w will be fixed. We define another ordering, this time we order the remaining vertices of U'_w according to the r-order of the rightmost adjacent vertex from V'_w (vertices with the lowest r-order value coming first). We call this ordering an *c-ordering*. Similarly to the case of the r-ordering, the c-ordering will be fixed.

Let us recap the current state. We have a bipartite graph where the part V'_w

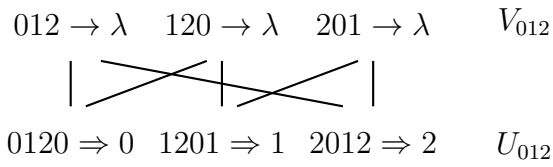


Figure 5.7: A configuration where each r-ordering of V_{012} leads to a false gap.

is ordered by an r-ordering and the part U'_w is ordered by a c-ordering. During the search for an optimal solution, we will remove some vertices from the original graph. It will be useful to preserve the following invariants:

- The current graph contains no false gaps.
- The remaining vertices from U'_w are ordered by a c-ordering.

Let U'_w have n vertices. We will describe how to find an optimal solution.

- For $n = 1$, it suffices to select any member of V'_w as a solution.
- For $n > 1$, let $u_0 \in U'_w$ be the smallest vertex in the c-ordering (if there are several equivalent vertices, we choose one arbitrarily), and let $v_1 \in V'_w$ be the greatest vertex (according to the r-ordering) adjacent to u_0. The vertex v_1 becomes a member of the solution. We remove u_0, v_1, and all neighbours and predecessors (in the r-ordering) of v_1 from the current graph (let us denote them UV_0). The remaining graph satisfies the invariant (as we prove below) and thus the smaller problem can be solved recursively. Then we add v_1 to the solution obtained for the smaller graph to form the solution for the case of whole U'_w.

It is important to note the following facts:

- It is safe to remove the predecessors of v_1 as all vertices adjacent to them are surely adjacent to v_1 as well (because there are no gaps and no false gaps). Therefore, those vertices from U'_w are covered by v_1.
- The graph obtained by removing vertices from UV_0 contains no gap (a gap can not be introduced as together with v_1 and its predecessors, we remove also all vertices from U'_w that could be part of the gap) and no false gap (otherwise, the original graph would obviously contain a gap or a false gap).
- It still holds that remaining vertices from U'_w are c-ordered, i.e. they are ordered according to the r-index of the greatest adjacent vertex in the smaller graph. This can be seen as follows. If the greatest adjacent vertex of a vertex $u_\mathrm{r} \in U'_w \setminus UV_0$ changed, then before removing UV_0 the greatest adjacent vertex was a member of UV_0. Thus, it was obviously the vertex v_1 or one of its predecessors, and as we already said above, this means that u_r is covered by v_1 and thus it also belongs to UV_0. This gives a contradiction. Thus, the greatest adjacent vertices of all vertices from $U'_w \setminus UV_0$ remain intact and thus the current ordering is a c-ordering.

It remains to show that this yields the optimal solution for the whole graph $G' = (U'_w \cup V'_w, E')$, where $|U'_w| = n$. For $n = 1$, we surely have the optimal solution. Let $n > 1$ be the smallest n such that our solution is not optimal. Let our solution $V'_{w,\mathrm{opt}}$ contain n' vertices. After removing vertices called UV_0 above, we obtain a graph for which our algorithm is supposed to return an optimal solution. It is obvious that the solution for this smaller graph contains exactly $n' - 1$ vertices. Let there be a better solution $V'_{w,\mathrm{better}}$ for G' containing at most $n' - 1$ vertices. We again remove the vertices from UV_0. As u_0 was surely covered by one of the vertices from $UV_0 \cap V'_{w,\mathrm{better}}$, we know that now there remain at

most $n' - 2$ vertices from $V'_{w,\text{better}}$ that cover $U'_w \setminus UV_0$. This is a contradiction as we supposed that our solution containing $n' - 1$ vertices is optimal. Thus, we have an optimal solution for the whole original graph G'.

We perform this process for all possible r-orderings of V_w and choose the solution having the smallest number of vertices picked to cover reductions. Then we create rewriting meta-instructions for each rewrite from the solution as described above. Together with rewriting meta-instructions with non-empty replacement word defined above and with accepting meta-instructions defined above as well, we have the complete set of meta-instructions that will be used in the output automaton. It is easy to see that the automaton will satisfy the requirements of this theorem.

Below we summarize the algorithm proposed in this proof (the set of accepting meta-instructions and the set of rewriting meta-instructions of the resulting automaton will be denoted *AMIS* and *RWIS*, respectively).

Algorithm 5.2.9.
Input: a type of restating automata $\mathsf{X} \in \{\mathsf{RRWW}, \mathsf{RRW}, \mathsf{RR}\}$, a working alphabet Γ, an input alphabet Σ, a positive $\overline{\text{R}}$-presentation $S = (S^+, \emptyset, R^+, \emptyset)$ of an unknown target restarting automaton over Γ, an upper bound $k \in \mathbb{N} \cup \{\infty\}$ on the size of the window.
Output: an X-automaton $M = (\Sigma, \Gamma, I)$ having the size of the window at most k, consistent with S, such that $\mathsf{isize}(M)$ is the smallest among all such automata.

if $S^+ = \emptyset$ **then**
    Let $AMIS = \emptyset$.
else
    Let $AMIS = (\Gamma^*, \text{Accept})$.
if $R^+ = \emptyset$ **then**
    Let $RWIS = \emptyset$.
else
    begin
        Let $R_0^+ \subseteq R^+$ be a set of reductions $x \Rightarrow x'$ such that x' can be obtained from x by removing its subword.

        Let $R_1^+ = R^+ \setminus R_0^+$.

        If R_1^+ contains a reduction that can not be performed by an X-automaton having the size of the window at most k, the problem has no solution.

        Let $G = (U \cup V, E)$ be a bipartite graph such that $U = R_0^+$ and V is a set of all pairs $(w' \to \lambda)$ such that w' is a rotation of a word w such that there is a reduction $(x \Rightarrow x') \in R_0^+$ that can be performed by removing w from x, and there is an edge between $(x \Rightarrow x') \in U$ and $(w \to \lambda) \in V$ if the reduction can be performed by replacing w with λ (if w longer than k symbols is needed, then the problem has no solution).

        Partition V so that classes are formed by rotations of the same

word. Let the individual classes be denoted with V_w (a class of rotations of w).

Let U_w be the set of all neighbours of vertices from V_w.

for all classes $V_w \subseteq V$ **do**
begin
Let $PS_{\min}$ be the smallest set PS of meta-instructions over all r-orderings of V_w where PS is obtained below.
begin
Let $v_0 = (w' \to \lambda)$ be the smallest vertex from V_w

Let $U'_w = \{u \in U_w; u \text{ is not adjacent to } v_0\}$.
Let $V'_w = V_w \setminus \{v_0\}$.
Let $E' = \{\{u, v\} \in E; u \in U'_w, v \in V'_w\}$.

Let $G' = (U'_w \cup V'_w, E')$.

Order U'_w into a c-ordering.

Let $PS = \{(\Gamma^*, w' \to \lambda, \Gamma^*)\} \cup \text{solve}(G')$, where the function solve is described below.
end

Add meta-instructions from $PS_{\min}$ to *RWIS*.
end

For each reduction $(x \Rightarrow x') \in R_1^+$ that can be performed by $i = (\Gamma^*, y \to y', \Gamma^*)$ (such that y and y' share neither a common non-empty prefix nor a common non-empty suffix), insert i into *RWIS*.
end

return $M = (\Sigma, \Gamma, AMIS \cup RWIS)$.

function solve($G' = (U' \cup V', E')$)
begin
if $|U'| = 1$ **then**
begin
Choose a rewrite $(y \to \lambda) \in V'$ at random.
return $\{(\Gamma^*, y \to \lambda, \Gamma^*)\}$.
end

Let u_0 be the smallest vertex from U' according to the c-ordering.
Let $v_1 = (y \to \lambda)$ be the greatest (according to the r-ordering) neighbour of u_0.

Let $U'' = \{u \in U'; u \text{ is not adjacent to } v_1\}$.

Let $V'' = \{v \in V'; v > v_1 \text{ in the r-ordering}\}$.
Let $E'' = \{\{u, v\} \in E'; u \in U''_w, v \in V''_w\}$.

Let $G'' = (U'' \cup V'', E'')$.
return $\{(\Gamma^*, y \to \lambda, \Gamma^*)\} \cup \text{solve}(G'')$.
end

The time complexity of this method is polynomial in the length of the input:

- Let the length of an input be n. Thus, n is an upper bound for the size of R^+ as well as for the length of every word present in reductions in R^+.
- Let us have a sliding reduction $r = (u \Rightarrow v) \in R^+$. Collecting all located reductions corresponding to r requires number of steps polynomial in n (there are at most n places where a rewriting can occur and the length of a rewrite is limited by n as well; moreover, to check that a particular pair of a place and the length of a rewritten subword leads to a located reduction corresponding to r requires at most n steps, too).
- To obtain R_0^+, we just select reductions that can be performed by a located reduction with the empty replacement word from the located reductions created as described above. Thus, it can be created in time polynomial in n. The set R_1^+ is formed by the remaining members of R^+.
- To create G and to partition V, we can proceed as follows. For each $r \in R_0^+$ that can be performed by $(x, w \to \lambda, y)$ (for some $x, w, y \in \Gamma^*$), we compute all rotations of w and remember the smallest one according to the lexicographical ordering. It is then easy to obtain partitioned V and to create G in time polynomial in n.
- The cycle over classes of the partition of V runs at most n times. There are at most n r-orderings of each V_w, thus the cycle searching for the minimal set PS runs at most n times. In the body of the cycle, after some initial steps (requiring obviously only time polynomial in n), the function solve is called. This function calls itself recursively, each time running on a smaller graph. As each run is polynomial in n (not counting the recursive call to the function solve) and the recursion has depth at most n (because each time a vertex from U_w is removed from the considered graph), we conclude that the time required by one run of the cycle is polynomial in n as well.
- Overall, the Algorithm 5.2.9 runs in time polynomial in n.

This concludes our proof for the isize measure. But note that the same method can be used in the case of size. Obviously, the solution for the size case can not have less meta-instructions. But we may ask if it is possible to somehow reach lower size complexity by picking suitable rewritten and replacement words (note that the contexts of meta-instructions can not have smaller size complexity as well as the accepting meta-instruction). Let us analyze our optimal solution for the isize case. If a rewriting meta-instruction is needed that uses a non-empty replacement word in order to perform a positive reduction, then it holds even in

the size case and the same meta-instruction must be used. On the other hand, let us consider the case where a rewriting meta-instruction rewriting a subword w to the empty word can be used in order to perform a positive reduction. Let us consider V_w in the run or our algorithm. The optimal solution for the isize case can not cover U_w with more vertices from V_w than the optimal solution for the size case. Moreover, the optimal solution for the size case can use only rewrites from V_w to cover U_w. Now note that all rewritten words present in vertices of V_w have the same lengths. Thus, the optimal solution for the size case has the same size complexity as our optimal solution for the isize case.

□

5.3 Complexity Summary

We determined the complexity of several restarting automata learning problems. In the case of learning from both positive and negative samples, we obtained rather negative results. However, those results form an important part of the research of the problem. Particularly, it gives a reason for using heuristic or time-consuming algorithms to solve those problems.

On the other hand, the case of learning from positive samples only was shown to be solvable effectively. Unfortunately, the resulting automata obtained using the proposed method can be considered unusable for nearly all cases in practice (though our results could be partially used to deal with sliding reductions if e.g. minimizing the number of meta-instructions is needed). Therefore, we conclude that there is a need for another way to learn restarting automata from positive samples only.

6. Proposed Method

This chapter presents our method for inferring models of the ABR. In Section 6.1 we present a similar method by Mráz, Otto and Plátek [55]. Our own method is proposed in Section 6.2.

6.1 Closely Related Work

As noted already in Section 2.2, our method is closely related to that one presented in [55]. Mráz, Otto and Plátek in [55] consider learning of a model of the ABR from positive samples as follows (they learn a model that is a restricted version of RRWW-automata):

- The input consists of sample *simple* positive words and sample positive reductions (note that they are not expected to be reductions of words from the target language, i.e. the input samples can contain reductions of words outside the target language, too).
- They assume a helpful teacher that partitions input samples into several groups to be handled by individual meta-instructions. Therefore, they do not have to deal with deciding what samples to use for learning particular meta-instructions, and they can focus on learning left and right contexts of a single meta-instruction at a time and on learning the base language. This reduces the task of inferring target language (that need not to be a regular language) to inferring only the languages used to describe how to parse words, i.e. to inferring at most regular languages. This is a problem that has been studied for several tens of years. An important thing to note here is that those languages can be easy to learn despite the complexity of the whole target language.
- They assume a helpful teacher that supplies *located* reductions only. This further simplifies inference of contexts of meta-instructions as there is no risk of a confusion caused by picking the wrong place.

The output of their algorithm is a special kind of a restarting automaton called strictly locally testable restarting automaton defined below.

Note the similarity to the $(\alpha, \mathsf{RRWW}, k, \Sigma, \Gamma, \dot{\mathrm{R}})$-inference optimization problem from positive data. We can consider a constant function α (thus, we are only searching for an automaton consistent with the input data) and let k, Σ, and Γ be as needed. However, the problems are not the same. Particularly, in [55] a more helpful teacher partitions input samples among individual meta-instructions that will handle them.

The automata used as models in [55] are based on strictly locally testable languages [73, 42] that are known to be learnable in the limit from positive data. This is extensively used in the inference method in [55], where the strictly locally testable languages are used in meta-instructions. This restricts the class of considered models. Let us start with the definition of the strictly locally testable languages:

Definition 6.1.1 ([73, 42]). *Let $k > 0$ and let Σ be an alphabet. A language $L \subseteq \Sigma^*$ is* strictly k-testable *if there exist finite sets $A, B, C \subseteq \Sigma^k$ such that for each $w \in \Sigma^{\geq k}$, it holds $w \in L$ if and only if*

- $\mathrm{P}_k(w) \in A$,
- $\mathrm{S}_k(w) \in B$, *and*
- $\mathrm{I}_k(w) \subseteq C$,

where $\mathrm{P}_k(w), \mathrm{S}_k(w)$, and $\mathrm{I}_k(w)$ denote k-length prefix of w, k-length suffix of w, and the set of all interior subwords of the length k of w, respectively. The (A, B, C) is called a triple *for L.*

A language L is called strictly locally testable *if L is strictly k-testable for some k.*

Now we can define strictly locally testable restarting automata. Compared to general restarting automata, they allow only languages from a proper subclass of regular languages to be used in meta-instructions, namely the strictly locally testable languages.

Definition 6.1.2 ([55]). *Let k be a positive integer and let Γ be an alphabet.*

A rewriting meta-instruction $(E_\ell, x \to y, E_\mathrm{r})$ is strictly k-testable*, if both E_ℓ and E_r are strictly k-testable and $|x| \leq k$.*

An accepting meta-instruction (E, Accept) is strictly k-testable *if E is strictly k-testable.*

A restarting automaton M is strictly k-testable*, if all its meta-instructions are strictly k-testable; we denote the class of such automata with k-*SLT-R*.*

A restarting automaton is strictly locally testable*, if it is strictly k-testable for some $k \geq 1$; we denote the class of such automata with* SLT-R*.*

Having an algorithm for inferring strictly k-testable languages, it is then straightforward how to learn k-SLT-R-automata (the method is taken from [55]). Thanks to a helpful teacher, the only thing one needs to do is to run that algorithm on particular words supplied by the teacher. The teacher creates a template for a restarting automaton — he gives the number of rewriting meta-instructions, supplies pairs of rewritten and replacement words, and also gives the appropriate number of accepting meta-instructions. The only missing things are the languages used as contexts and as base languages. Each of those languages is determined individually by running an inference algorithm on a set of sample words supplied by the teacher. After that, we obtain the resulting automaton.

6.2 Our Approach

We will use a different approach. At first, we learn from FPABR samples (this also means that reductions of words outside the target language can not be present in the input samples). Samples of simple words are just the last elements of those sample analyses. Of course, this approach could be also used in the above

mentioned method for learning k-SLT-R automata after adding the remaining requested information (the number of rewriting meta-instructions, etc.). Our use of FPABR samples as input is rather related to the environment where we will evaluate our method later. Our method can be easily adjusted to learn from individual sample reductions and sample words. However, we like the idea of learning from FPABR samples as this ensures a nice property that all words present in the input samples will be accepted by the inferred automaton because our method always returns an automaton performing all presented reductions and accepting all given simple words.

Even if we know how to perform the ABR for some unknown target language, it may not be clear whether some reductions should be performed using the same meta-instruction in some unknown underlying restarting automaton we are looking for. Our approach does not expect the teacher to partition reductions into separated groups handled by different meta-instructions. Instead, we use a very simple approach — we use a heuristics that all reductions performing the same rewrite are handled by the same meta-instruction. This may seem to be quite limiting, but this way, we need not a helpful teacher, and more importantly, the achieved results are very good (as presented in Chapter 7).

We also do not assume a teacher locating rewritings in reductions. At first, we believe that the located reductions need not be always obvious (e.g. if the teacher does not know the target language well — it is possible that the teacher knows just the samples where the reductions are not located). Moreover, we will show that sometimes we need not to take care about the actual position at all. Though it may not be the case for the model considered in our approach, we consider our method to be a step on a long way towards learning more powerful models where this idea will be well applicable. And again, our approach to this problem is quite straightforward — we always consider the shortest rewritings (i.e. rewriting as short subword as possible) and among them the left-most one.

Our learning method works with a limited version of RRWW-automata in a similar way as in [55]. Instead of locally testable languages, we utilize reversible languages [4, 51] as the languages used in left and right contexts of meta-instructions and in accepting meta-instructions. According to [42], the class of reversible languages is a class greater than the class of locally testable languages. Details will be presented later.

We can briefly outline our method as follows:

1. Obtain individual located reductions from the input (remember we obtain sliding reductions in the input samples, but we transform them into located ones).

2. Associate the located reductions with particular rewriting meta-instructions that will handle them.

3. Learn rewriting meta-instructions using the located reductions associated in the previous step.

4. Obtain simple words from the input samples.

5. Learn exactly one accepting meta-instruction.

Though our method starts with sliding reductions, in order to learn rewriting meta-instructions, it needs located ones. As already mentioned, sometimes we can ignore actual places of rewritings. We study this idea in Subsection 6.2.1.

6.2.1 Reduction Selection

In Section 2.2 we defined two kinds of reductions: the located and the sliding ones. It is often easier for the teacher to give the sliding reductions. On the other hand, it can be easier to learn from the located ones for the learner. Though there is a simple way to transform a sliding reduction to a located one by considering e.g. the shortest possible rewritings and among them the left-most one, an important question arises: Does this approach limit our ability to learn some languages? In other words, could considering one particular located version of a reduction instead of another one prevent the inference of the target language?

One may ask whether it is necessary to distinguish those two cases of specifying input instead of considering the sliding one only and then picking some particular located reduction automatically. If we are interested in the inference of the target language and not in the exact inference of the particular ABR of the target language, then we maybe even do not care about the exact place where reductions occur. Or, we can be interested in the reduction relation without exact locations of rewritings — so we can consider different locations of changes to be equivalent. This problem is further analyzed in this subsection.

From Theorem 5.2.7 we know how a pair of rewritten and replacement subwords (i.e. a rewrite) $v \to v'$ where $(v \in \Gamma^+, v' \in \Gamma^*)$ looks like if there is a word where such rewriting can be deferred by one symbol.

Definition 6.2.1. *A rewrite $v \to v'$ is* deferrable *if either $v' = \lambda$ or $v'_0 = v_0$.*

Note that if a rewrite is deferrable according to the above presented definition, it does not mean that a corresponding rewriting can be deferred in every case. For example, the rewriting in the located reduction $(\lambda, 0 \to \lambda, \lambda)$ obviously cannot be deferred despite the fact that the replacement word is empty and thus, according to our definition above, the rewrite $0 \to \lambda$ is called deferrable.

To make further work easier, let us limit possible rewrites using the following theorem.

Theorem 6.2.2. *For each* RRWW*-automaton M, there is an* RRWW*-automaton M' such that all meta-instructions of M' performing deferrable rewrites are those rewriting to λ, and $\mathrm{L}(M) = \mathrm{L}(M')$.*

Proof. We transform each rewriting meta-instruction performing a deferrable rewrite using a non-empty replacement word. It has form of $(L_\ell, av \to av', L_\mathrm{r})$ according to the Definition 6.2.1. It can be replaced with $(L_\ell \cdot \{a\}, v \to v', L_\mathrm{r})$ while preserving the language accepted by the original automaton. Note that it may be necessary to repeat this transformation several times, until a non-deferrable rewrite or a rewrite to the empty word is obtained.

□

Now we can suppose that all rewriting meta-instructions performing a deferrable rewrite use the empty replacement word.

Theorem 6.2.3. *For each* RRWW*-automaton M, there is an* RRWW*-automaton M' such that:*

- $\mathrm{L}(M') = \mathrm{L}(M)$, *and*
- *no rewriting performed by M' can be deferred.*

Proof. Let $M = (\Sigma, \Gamma, I)$. According to Theorem 6.2.2, we can suppose that all meta-instructions of M performing deferrable rewrites rewrite to λ. We will show how to modify individual rewriting meta-instructions so that no rewriting they perform can be deferred. We will construct a new RRWW-automaton $M' = (\Sigma, \Gamma, I')$ as follows. Let $m = (L_\ell, w \to \lambda, L_\mathrm{r}) \in I$ be a rewriting meta-instruction.

Let $\mathrm{pr}(w, i)$ denote the prefix of w of the length i.

For the meta-instruction m, we define the following new meta-instructions and place them into I':

- $\mathrm{m}(i) = (L_\ell \cdot \mathrm{pr}(w, i), w' \to \lambda, \mathrm{LQ}(L_\mathrm{r}, \mathrm{pr}(w, i)) \setminus (\{w'_0\} \cdot \Gamma^*))$ for $0 \leq i < |w|$ and $w' = w \lll i$, and
- $\mathrm{m}(i, X) = (L_\ell \cdot \{w\} \cdot X, w' \to \lambda, \mathrm{LQ}(L_\mathrm{r}, X \cdot w') \setminus (\{w'_0\} \cdot \Gamma^*))$ for all $0 \leq i < |w|$, $w' = w \lll i$ (i.e. a rotation of w by i symbols), and for all maximal sets $X \subseteq \{w\}^* \cdot \{\mathrm{pr}(w, i)\}$ such that they do not contain members of more than one class of congruence (according to the Nerode's theorem) given by the language L_r. Let us denote the set of all such X as $X_{m,i}$.

To help you understand how $\mathrm{m}(i, X)$ looks like, we present an example. Let $m = (\lambda, 01 \to \lambda, \{01, 0101\})$. We will construct $\mathrm{m}(0, X)$. At first, we need the set $X_{m,0}$. It contains subsets of $\{w\}^* \cdot \{\mathrm{pr}(w, 0)\} = \{01\}^* \cdot \lambda = \{01\}^*$. The equivalence given by the Nerode's theorem contains the following classes: $\{\lambda\}$, $\{0\}$, $\{01\}$, $\{010\}$, $\{0101\}$, and $\{0, 1\}^* \setminus \{\lambda, 0, 01, 010, 0101\}$, let them form the set X_c. Then, $X_{m,0} = \{C \cap (\{w\}^* \cdot \{\mathrm{pr}(w, 0)\}); C \in X_\mathrm{c}\} = \{C \cap \{01\}^*; C \in X_\mathrm{c}\}$ $= \{\emptyset, \{\lambda\}, \{01\}, \{0101\}, \{(01)^p; p \geq 3\}\}$. Then we have:

- $\mathrm{m}(0, \emptyset)$ is a rewriting meta-instruction having the empty left context (because of the concatenation with $X = \emptyset$), so we can safely omit this meta-instruction.
- $\mathrm{m}(0, \{\lambda\}) = (\{\lambda\} \cdot \{01\} \cdot \{\lambda\}, 01 \to \lambda, \mathrm{LQ}(\{01, 0101\}, \{\lambda\} \cdot \{01\}) \setminus (\{0\} \cdot \{0, 1\}^*)) = (01, 01 \to \lambda, \lambda)$.
- $\mathrm{m}(0, \{01\}) = (\{\lambda\} \cdot \{01\} \cdot \{01\}, 01 \to \lambda, \mathrm{LQ}(\{01, 0101\}, \{01\} \cdot \{01\}) \setminus (\{0\} \cdot \{0, 1\}^*)) = (0101, 01 \to \lambda, \lambda)$.
- $\mathrm{m}(0, \{0101\})$ is a rewriting meta-instruction with the right context equal to $\mathrm{LQ}(\{01, 0101\}, \{0101\} \cdot \{01\}) \setminus (\{0\} \cdot \{0, 1\}^*) = \emptyset$, so we can safely omit this meta-instruction.
- $\mathrm{m}(0, \{(01)^p; p \geq 3\})$ is a rewriting meta-instruction with the right context equal to $\mathrm{LQ}(\{01, 0101\}, \{(01)^p; p \geq 3\} \cdot \{01\}) \setminus (\{0\} \cdot \{0, 1\}^*) = \emptyset$, so we can safely omit this meta-instruction as well.

Note that, in total, these meta-instructions perform exactly the same (sliding) reductions as the original meta-instruction m.

It is important to note that the contexts of the newly created meta-instructions are regular languages (use Theorem 1.1.20).

The approach is based on the following ideas:

- No deferrable rewrite can be performed prematurely. This is easily achieved by restricting the right contexts of the new meta-instructions to words not starting with the first symbol of the rewritten subword.
- All rewritings corresponding to deferrable rewrites performed by m must be deferred as far as possible and eventually performed by some $\mathrm{m}(i)$ or $\mathrm{m}(i, X)$ at the rightmost possible position in the word being reduced.

We will discuss the latter requirement now.

Let us start with the case when a rewriting can be deferred by at least $|w|$ symbols. Let $x, y, z \in \Gamma^*, i \in \{0, \ldots, |w| - 1\}$ (and let w' denote $w \lll i$) such that

- $(x, w \to \lambda, yw'z) \in m$, i.e. m can be used to directly reduce $xwyw'z$ to $xyw'z$,
- $xyw'z = xwyz$, i.e. by removing either w or w' from $xwyw'z$, we obtain the same word, and
- $z = \lambda$ or $z_0 \neq w'_0$, i.e. the rewriting consisting in removing w' can not be further deferred.

Obviously, the set of all words $xwyw'z$ fulfilling the above requirements is equal to $L_\ell \cdot \{w\} \cdot (L_\mathrm{r} \cap \{w\}^* \cdot \{\mathrm{pr}(w, i) \cdot w'\} \cdot (\Gamma^* \setminus (\{w'_0\} \cdot \Gamma^*)))$.

The set of words reduced by $\mathrm{m}(i, X)$ is equal to $L_\ell \cdot \{w\} \cdot X \cdot w' \cdot (\mathrm{LQ}(L_\mathrm{r}, X \cdot w') \setminus (\{w'_0\} \cdot \Gamma^*))$. It is important to remember that $X \in X_{m,i}$ was chosen in such a way that it fits into exactly one equivalence class in the congruence. Hence, for any $u \in X$, it holds $\mathrm{LQ}(L_\mathrm{r}, X \cdot \{w'\}) = \mathrm{LQ}(L_\mathrm{r}, uw')$. Therefore, $\bigcup_{X \in X_{m,i}} X \cdot w' \cdot (\mathrm{LQ}(L_\mathrm{r}, X \cdot w') \setminus (\{w'_0\} \cdot \Gamma^*)) = L_\mathrm{r} \cap \{w\}^* \cdot \{\mathrm{pr}(w, i) \cdot w'\} \cdot (\Gamma^* \setminus (\{w'_0\} \cdot \Gamma^*))$. Then the set of all words reduced by all $\mathrm{m}(i, X)$ for all X is equal to $L_\ell \cdot \{w\} \cdot (L_\mathrm{r} \cap \{w\}^* \cdot \{\mathrm{pr}(w, i) \cdot w'\} \cdot (\Gamma^* \setminus (\{w'_0\} \cdot \Gamma^*)))$. So $\mathrm{m}(i, X)$ meta-instructions handle all reductions performed by m on words satisfying the conditions — we know the same words are directly reduced in both cases, and by inspecting $\mathrm{m}(i, X)$, we see that the same words are obtained when the meta-instruction $\mathrm{m}(i, X)$ is applied as in the case when m is applied.

It remains to discuss the case when it is possible to defer a rewriting by at most $i \leq |w| - 1$ symbols. Let $u, y \in \Gamma^*$ such that

- $(u, w \to \lambda, \mathrm{pr}(w, i)y) \in m$, (please note that by removing either w or $w_i \ldots w_{|w|-1}\mathrm{pr}(w, i)$, we obtain the same word), and
- $y = \lambda$ or $y_0 \neq w_i$, i.e. the rewriting consisting in removing the word $w_i \ldots w_{|w|-1}\mathrm{pr}(w, i)$ can not be further deferred.

Obviously, the set of all words $uw\mathrm{pr}(w,i)y$ fulfilling the above requirements is equal to $L_\ell \cdot \{w\} \cdot (L_\mathrm{r} \cap (\{\mathrm{pr}(w,i)\} \cdot (\Gamma^* \setminus (\{w_i\} \cdot \Gamma^*))))$.

The set of words reduced by $\mathrm{m}(i)$ is equal to $L_\ell \cdot \{\mathrm{pr}(w,i)\} \cdot \{w \lll i\} \cdot (\mathrm{LQ}(L_\mathrm{r}, \mathrm{pr}(w,i)) \setminus (\{w_i\} \cdot \Gamma^*)) = L_\ell \cdot \{w\} \cdot (L_\mathrm{r} \cap (\{\mathrm{pr}(w,i)\} \cdot (\Gamma^* \setminus (\{w_i\} \cdot \Gamma^*))))$. Thus, the sets of reduced words are the same again. And as before, the words obtained by applying $\mathrm{m}(i)$ are obviously the same as when the original meta-instruction m is applied.

This way, we transformed every rewriting meta-instruction m that performs a deferrable rewrite into several new ones. The rewriting meta-instructions performing only non-deferrable rewrites are put into I' without any change. The same holds for accepting meta-instructions. The resulting restarting automaton will be $M' = (\Sigma, \Gamma, I')$. □

The automaton M' obtained from the theorem presented above performs all deferrable rewrites in such a way that the corresponding rewriting can not be further deferred. We could also say that the reduction occurred at the rightmost possible place. Note that we can easily use that theorem to prove a similar theorem for the case of reductions occurring at the leftmost possible places.

Theorem 6.2.4. *For each* RRWW*-automaton M, there is an* RRWW*-automaton M' such that:*

- $\mathrm{L}(M') = \mathrm{L}(M)$, *and*
- *no rewriting performed by M' is a deferred rewriting.*

Proof. Having M represented using meta-instructions, it is quite easy to transform it into M^R such that $\mathrm{L}(M^\mathrm{R}) = (\mathrm{L}(M))^\mathrm{R}$ and the working of M^R exactly mirrors the working of M. Then using Theorem 6.2.3, we obtain M_r^R performing rewritings that can not be deferred. Let us now revert M_r^R in the same way as we did before with M. We obtain M' satisfying the requirements of the theorem. □

As we said above, we do not expect a teacher able to supply located reductions. Thanks to the analysis of deferrable rewrites we performed above, we see that it is somehow safe to deal with sliding reductions as with the located ones where the rewriting of the shortest possible subword occurs at the leftmost possible position — we will call such rewritings the first-opportunity rewritings.

Definition 6.2.5. *Let $\bar{\mathrm{r}} = (u \Rightarrow v)$ be a sliding reduction ($u, v \in \Sigma^*$ for some alphabet Σ). Let $\dot{\mathrm{X}} = \{(x, y \to y', z); xyz = u, xy'z = v,$ where $x, y, y', z \in \Sigma^*\}$ be a set of all located reductions that also transform u to v. We associate the value $(|y|, |x|)$ with each located reduction $(x, y \to y', z) \in \dot{\mathrm{X}}$. Then we order the set $\dot{\mathrm{X}}$ according to this value lexicographically. The rewriting performed by the smallest reduction in this order is called the* first-opportunity rewriting.

However, remember that we do not work with the general RRWW-automata in our approach — proposed restrictions will be introduced later in this thesis. The question whether similar results on deferrable rewrites hold in this restricted case is still an open problem.

6.2.2 Search Space

Our search space is a class of restricted RRWW-automata we will introduce in this subsection. Similarly to the approach presented in [55] and described in Section 6.1, our one is based on a subclass of regular languages that is learnable from positive data only. We decided to use zero-reversible languages [4, 51] defined below.

Definition 6.2.6 ([4, 51])**.** *A finite state automaton A is* zero-reversible *if both A and A^{R} are deterministic. A language L is* zero-reversible *if there is a zero-reversible finite state automaton A such that $L = \mathrm{L}(A)$.*

We will use zero-reversible languages both in contexts of rewriting meta-instructions and in accepting meta-instructions. Our method first collects positive samples of particular languages (e.g. samples from which to learn the left context of some rewriting meta-instruction) and then we infer that language from those samples. To perform this inference, we use the algorithm ZR [4], taking a finite nonempty set of words S and returning a DFSA A such that $\mathrm{L}(A)$ is the smallest zero-reversible language containing S. In [4] it is shown that the algorithm ZR may be used to identify zero-reversible languages in the limit. We will describe it below. The code is taken from [4]. We will work with a partition of the set of states of a FSA. The values $\mathrm{s}(B, b)$ and $\mathrm{p}(B, b)$ denote the b-successors and b-predecessors of a block B in the current partition of the set of states. $\mathrm{B}(q, \pi)$ denotes the procedure which returns the block of the partition π containing the element q.

Algorithm 6.2.7. Algorithm ZR.
Input: a nonempty positive sample S.
Output: a zero-reversible finite state automaton A accepting all words from S.

Let $A_0 = (Q_0, \Sigma, \delta_0, I_0, F_0)$ be $\mathrm{PTA}(S)$.
Let π_0 be the trivial partition of Q_0.
For each $b \in \Sigma$ and $q \in Q_0$, let $\mathrm{s}(\{q\}, b) = \delta_0(q, b)$.
For each $b \in \Sigma$ and $q \in Q_0$, let $\mathrm{p}(\{q\}, b) = \delta_0^{\mathrm{R}}(q, b)$.
Choose some $q' \in F_0$.
Let $LIST$ contain all pairs (q', q) such that $q \in F_0 \setminus \{q'\}$.
Let $i = 0$.
while $LIST \neq \emptyset$ **do**
begin
    Remove some element (q_1, q_2) from $LIST$.
    Let $B_1 = \mathrm{B}(q_1, \pi_i), B_2 = \mathrm{B}(q_2, \pi_i)$.
    if $B_1 \neq B_2$ **then**
    begin
        Let π_{i+1} be π_i with B_1 and B_2 merged.
        For each $b \in \Sigma$ do $\mathrm{sUPDATE}(B_1, B_2, b)$.
        For each $b \in \Sigma$ do $\mathrm{pUPDATE}(B_1, B_2, b)$.
        Increase i by 1.
    end
end

return the finite state automaton A_0/π_i.

The procedure named sUPDATE(B_1, B_2, b) places $(\mathrm{s}(B_1, b), \mathrm{s}(B_2, b))$ on the $LIST$ if both $\mathrm{s}(B_1, b)$ and $\mathrm{s}(B_2, b)$ are nonempty, and defines $\mathrm{s}(B_3, b)$ to be $\mathrm{s}(B_1, b)$ if this is nonempty and $\mathrm{s}(B_2, b)$ otherwise (where B_3 is the union of B_1 and B_2). The procedure pUPDATE is defined similarly, with p in place of s.

In what follows, we will write $\mathsf{ZR}(S)$ to denote the output of the algorithm ZR on the input S.

It is easy to see that the transition function of the resulting automaton need not be total — if it is the case, then the automaton is not complete but just deterministic. For example, let us suppose $S = \{01\}$. Then the initial finite state automaton $A_0 = \mathrm{PTA}(S)$ in the run of the ZR-algorithm contains exactly one accepting state. Therefore, the while-cycle is never entered and A_0 is returned. And, of course, δ_0 is not total as e.g. $\delta_0(i_0, 1) = \emptyset$ where $I_0 = \{i_0\}$. But it is easy to transform the resulting automaton into a complete one by adding a non-accepting dead state (but note that adding that state could break the zero-reversibility of the automaton).

Having a class of languages and an algorithm for inferring them, we are ready to introduce the model considered in this thesis.

The languages allowed to appear in contexts of rewriting meta-instructions and as base languages in accepting meta-instructions will be only those from the class of zero-reversible languages. This way, we will be able to infer them once we have positive samples.

Moreover, it will be a feature of our learning method that, for all $u \in \Gamma^+, v \in \Gamma^*$ (Γ being the considered working alphabet), the resulting automata will have at most one rewriting meta-instruction performing a rewrite $u \to v$ (where u and v share neither a common non-empty prefix nor a common non-empty suffix) and also only one accepting meta-instruction. It is therefore natural to introduce this restriction into the model as well. It will help us better understand the power of the models returned by our method. The formal definition is below.

Definition 6.2.8. *A* rewriting meta-instruction $(E_\ell, x \to y, E_\mathrm{r})$ *is called* zero-reversible*, if both E_ℓ and E_r are zero-reversible languages.*

An accepting meta-instruction (E, Accept) *is called* zero-reversible*, if E is a zero-reversible language.*

A restarting automaton *is called* zero-reversible*, if all its meta-instructions are zero-reversible.*

A restarting automaton M *is called* single *(a* S-RRWW*-automaton), if the following requirements hold:*

(S1) M has at most one accepting meta-instruction.

(S2) For each pair $x \to y$, where $x, y \in \Gamma^*$, M has at most one meta-instruction of the form $(E_\ell, x \to y, E_\mathrm{r})$.

(S3) For each rewriting meta-instruction $(E_\ell, x \to y, E_\mathrm{r})$, it holds that x and y share neither a common non-empty prefix nor a common non-empty suffix.

A restarting automaton *is called* single zero-reversible *(a* S-ZR-RRWW*-automaton), if it is both single and zero-reversible. We denote the class of languages accepted by* S-ZR-RRWW*-automata by $\mathcal{L}$(S-ZR-RRWW).*

This model is very similar to that one we have used in [31]. There the last requirement (denoted with (S3)) for an automaton to be single from the above definition was not considered. However, omitting (S3) allows, in some extent, to bypass the requirement (S2), e.g. there can be rewriting meta-instructions such as

- $(L_1, 0 \to \lambda, L_2)$,
- $(L_3, 00 \to 0, L_4)$, and
- $(L_5, 000 \to 00, L_6)$,

that in some sense perform the same rewrite.

Because our method will never return automata violating (S3), we introduced this feature into the Definition 6.2.8 as well.

Note that there are more differences between S-ZR-RRWW-automata and SLT-R-automata than the classes of languages allowed in meta-instructions:

- We consider only *single* restarting automata.
- We have no limit on the length of the rewritten word in meta-instructions (in the case of k-SLT-R-automata, the length is limited by k).

In our inference approach, we partition the reductions presented in the input samples into several classes based on the performed rewrites $u \to v$, and then each class is used to infer a single rewriting meta-instruction. By restricting to single restarting automata, we completely avoid the problem of partitioning the reduction samples performing the same rewrite into groups corresponding to rewriting meta-instructions which differ in contexts only.

The following theorem characterizes the power of the discussed model. The proof is a modification of the proof of a similar result presented in [55].

Theorem 6.2.9. $\mathsf{GCSL} \subseteq \mathcal{L}(\mathsf{S\text{-}ZR\text{-}RRWW}) \subseteq \mathsf{CSL}$.

Proof. We will prove that $\mathsf{GCSL} \subseteq \mathcal{L}(\mathsf{S\text{-}ZR\text{-}RRWW})$ at first. Let $G = (\Sigma, \Gamma, S, P)$ be a growing context-sensitive grammar. We will construct a S-ZR-RRWW-automaton M such that $\mathrm{L}(M) = \mathrm{L}(G)$. W.l.o.g. we suppose that P does not contain any rules $S \to X$ for $X \in \Gamma$ (otherwise, we could replace each such rule with several rules $S \to \alpha$ for all α such that $X \to \alpha \in P$).

We will iteratively modify the original grammar, until we reach a form suitable for an easy transformation to a S-ZR-RRWW-automaton:

1. With each terminal symbol $a \in \Sigma$, we associate the new non-terminal symbol $\overline{a}$ (let the new symbols form the alphabet $\overline{\Sigma}$). This step will later help us to manipulate with "terminal" symbols during rewrites as needed.
2. For each rule $\alpha \to \beta \in P$, we create the set of new rules of the form $\alpha' \to \beta'$ (these new rules will be placed into $\overline{P}$) where
   - α' is obtained from α by replacing each occurrence of a terminal symbol a with the associated non-terminal symbol $\overline{a}$ (note that this means there is no terminal symbol on the left-hand side of the resulting rule), and

- β' is obtained from β by replacing *some* occurrences of terminal symbols a with the associated non-terminal symbol $\overline{a}$ in all possible ways (including replacing all of them and also none of them).

Let $\Gamma_1 = \Gamma \cup \overline{\Sigma}$ and let the resulting grammar be $G_1 = (\Sigma, \Gamma_1, S, \overline{P})$.

3. With each non-terminal symbol $X \in \Gamma_1$, we associate the new non-terminal symbol $\overline{X}$ (let those new non-terminal symbols form the alphabet $\overline{\Gamma_1}$).

4. For each rule $\alpha \to \beta \in \overline{P}$, we create the rules $\alpha' \to \beta'$ (these new rules will form P') where

   - α' is obtained from α by replacing all occurrences of non-terminal symbols X with the associated non-terminal symbols $\overline{X}$ in all possible ways (including replacing all of them and also none of them), and
   - β' is obtained from β by changing its first symbol and its last symbol to be different from the first occurrence of a non-terminal symbol in α and from the last occurrence of a non-terminal symbol in α, respectively, by replacing the particular occurrences of non-terminal symbols X in β' with the associated non-terminal symbols $\overline{X}$ (when necessary, i.e. if the first or last symbol of β is a terminal symbol or it is different from the first or the last symbol of α, then no change is performed).

   Let $\Gamma' = \Gamma_1 \cup \overline{\Gamma_1}$ and let the obtained grammar be $G' = (\Sigma, \Gamma', S, P')$.

Let us show that $\mathrm{L}(G) = \mathrm{L}(G')$. The first step just added some new non-terminal symbols. Note that they are used only in places where original terminal symbols can occur. Thus, if we ignore the difference between the original terminal symbols and the associated non-terminal symbols, the grammar G_1 rewrites in the same way as G. Moreover, once G_1 yields a word over Σ, this word thus surely belongs to $\mathrm{L}(G)$. On the other hand, let us have a derivation of some word $w \in \mathrm{L}(G)$. We go from the end of the derivation and look for places where particular occurrences of terminal symbols were written last time. We replace all the previous occurrences with associated non-terminal symbols. This obviously gives a derivation for w using the rules of G_1.

The third step just adds new non-terminal symbols that will be used in the fourth one. The idea of the transformation of G to $\mathsf{S\text{-}ZR\text{-}RRWW}$-automaton is to associate each rule $\alpha \to \beta$ with the rewriting meta-instruction $(Z^*, \beta \to \alpha, Z^*)$ (where Z denotes the working alphabet of the restarting automaton). However, e.g. in the case of the rule $0X0 \to 0110$ this idea would lead to the rewriting meta-instruction $(Z^*, 0110 \to 0X0, Z^*)$ that violates the condition for an automaton to be single (the condition (S3) of Definition 6.2.8). Unfortunately, in the case of zero-reversible restarting automata, the contexts of rewriting meta-instructions can be too weak for us to be able to convert this meta-instruction into $(Z^* \cdot \{0\}, 11 \to X, \{0\} \cdot Z^*)$ to avoid the mentioned violation. Therefore, we change the rewriting rules so that the left and right sides will both start and end with different symbols. This is exactly what is performed in the fourth step.

Again, does this preserve the generated language? As before, we see the rewriting performed by both grammars G_1 and G' is the same if we ignore the difference between the original symbols and the associated non-terminal symbols.

Particularly, if G' yields a word over Σ, it surely belongs to $\mathrm{L}(G_1)$. On the other hand, let us have a derivation for $w \in \mathrm{L}(G_1)$. Because of the changes made in the first two steps, we know that the terminal symbols occur only on the right-hand sides of rules used in the derivation. Note that for each rule used in the derivation, we have at least a similar rule that differs only in using some associated non-terminal symbols in place of the original ones. We start at the beginning of the derivation. We proceed iteratively, using rules of G' corresponding to the rules used originally (i.e. the only difference is the presence of associated non-terminal symbols in place of some original non-terminal symbols). We know the corresponding rule is always available in P' as our left sides cover all possible ways of using those associated non-terminal symbols. Eventually, there remain no non-terminal symbols. So, w is derived using G'.

Finally, we define $\Gamma'' = \Sigma \cup \Gamma'$ and $M = (\Sigma, \Gamma'', I)$, where:

- For each rule $\alpha \to \beta \in P'$ such that $|\alpha| < |\beta|$, there is $(\Gamma''^*, \beta \to \alpha, \Gamma''^*) \in I$.
- If $\lambda \notin \mathrm{L}(G)$, then $(\{S\} \cup \{a \in \Sigma; S \to a \in P'\}, \mathrm{Accept}) \in I$.
- If $\lambda \in \mathrm{L}(G)$, then
  - $(\{\lambda\}, \mathrm{Accept}) \in I$,
  - $(\{\lambda\}, S \to \lambda, \{\lambda\}) \in I$, and
  - $(\{\lambda\}, a \to \lambda, \{\lambda\}) \in I$ for all $a \in \Sigma$ such that $S \to a \in P'$.

It can be easily seen that M is a S-ZR-RRWW-automaton. The automaton M tries to find derivations of given words (with a few exceptions handled by the accepting meta-instruction) and accepts or rejects the input words accordingly. Thus, $\mathrm{L}(G) = \mathrm{L}(M)$.

The inclusion $\mathcal{L}(\text{S-ZR-RRWW}) \subseteq \mathsf{CSL}$ follows from the fact that each restarting automaton can be simulated by a linear bounded automaton.

□

We have a model and a method for inferring the languages used in the internals of that model. We would like to infer the model from some samples of the unknown target language. As already mentioned in Chapter 2, there are several approaches to the problem setting. We also said that our method is different in that it expects sample reductions. Actually, we go even further and ask for input consisting of FPABR samples, i.e. for a given word there is given also a sequence of words representing its ABR until a correct simple word is obtained — let us recall the definition of a FPABR sample (Definition 4.2.1).

Definition. A *fully positive analysis by reduction sample* (FPABR sample) for a language L is a finite sequence $\{w_i\}_{i=0}^{n}$, $w_i \in L$ for all $i \in \{0, \ldots, n\}$, and $|w_i| > |w_{i+1}|$ for all $i \in \{0, \ldots, n-1\}$.

The sample defined above represents a correct way of how to iteratively reduce a correct word to a correct simple word in order to check the initial word for correctness. The last member of the sequence is thought as a simple word, i.e. no further reducing of the last word is possible. Note that there is no relation to any particular model of the ABR in the definition, but quite naturally, we suppose some particular model of the ABR behind FPABR samples in practice.

We expect the input consisting of FPABR samples to improve the performance of learning algorithms as it gives complete ways of how to parse words instead of sample reductions that may not be related. Here just by remembering the input, we are able to fully parse at least the words present in the input samples.

In practice, there can be a teacher wanting to describe how to do the ABR of some natural language. This is not only hypothetic, the ABR forms an important tool for parsing e.g. Czech sentences. The teacher is able to show us a lot of samples of how to completely parse sentences using the ABR. This corresponds to a set of FPABR samples.

Below we summarize the main problem considered in this thesis. Note both the difference and the similarity to the problems analyzed in Chapter 5. There we considered a more common approach where both the positive and the negative input sample words were presented. The problem discussed in Chapter 5 was shown to be too hard. However, our ultimate goal is to learn from positive samples only and we are not so strict regarding the expected output (particularly, we do not require the smallest model). Let us recall the ABR inference from positive samples problem (Definition 4.2.2).

Definition. An *ABR inference from positive samples problem* is defined as follows:

- The input consists of a finite set of FPABR samples S, an input alphabet Σ, and a working alphabet Γ.
- The goal is to find an RRWW-automaton $M = (\Sigma, \Gamma, I)$ consistent with S, i.e. performing all the given reductions and accepting all the given simple words by some of its accepting meta-instructions. I.e. for each FPABR sample $\{w_i\}_{i=0}^{n} \in S$, it holds $w_0 \vdash_M w_1 \vdash_M \cdots \vdash_M w_n$ and $w_n \in E$ for some accepting meta-instruction $(E, \text{Accept}) \in I$.

Note the difference to the usual inference problem from Definition 2.1.3. We do not take as input only positive samples of words, though each given word belongs to the language. In addition, we have a reduction relation saying that one word can be reduced to another one. The side product of learning the ABR is the ability to accept the corresponding language.

Moreover, once we have a solution for the just defined problem, we know that the resulting automaton accepts also all the initial extended words present in the input as they will surely be reduced to simple words and then accepted.

It may not be obvious why to include the input and working alphabets in the input of the algorithm. The resulting restarting automaton must have the alphabets specified and, generally, it is not possible to infer them from the input samples only, so they are required to be in the input. But one can also ask why to allow any auxiliary symbols in the input. Our methods are motivated linguistically and in that area, auxiliary symbols like NP (for a noun phrase) or PP (for a prepositional phrase) are often used to describe structure of sentences. In other words, they are sometimes used by teachers to describe the rules underlying the language under examination. Thus, we allow them to appear in the input as well.

The next section presents a simple method for inferring S-ZR-RRWW-automata from FPABR samples.

6.3 The Omega2 method

This section describes our approach to solving the ABR inference from positive samples problem. At the beginning, we have to choose a model to infer. We decided to use S-ZR-RRWW-automata. The consequence of this decision is twofold.

At first, as we deal with single restarting automata, we know that all input reductions performing a rewrite $x \to y$ should be handled by the same rewriting meta-instruction of the inferred automaton. Similarly, all simple words present in the input samples should be accepted by the same accepting meta-instruction. In contrast, [55] assumes a very helpful teacher that assigns the input samples to particular meta-instructions to be inferred. It would be easy to modify our method to avoid the restriction on single restarting automata when such a teacher is available.

At second, as considered automata are zero-reversible, once we know the sample words belonging to contexts of a particular rewriting meta-instruction, we can infer the corresponding languages using the ZR algorithm. Using the ZR algorithm, we are assured that the contexts are the smallest possible ones. Once a teacher finds something is missing, he can extend the input samples and run the algorithm again. Similarly, once we have the set of samples of simple words, we can run the ZR algorithm to obtain an automaton representing the language to be placed into the only one accepting meta-instruction.

Let us now present the complete run of the method. The input samples will be transformed into individual located reductions using the first-opportunity rewritings and simple sentences will be collected. Then we split the located reductions among individual rewriting meta-instructions to be learnt based on the rewrite they perform. We call reductions performing the same rewrite *compatible reductions* as formalized below:

Definition 6.3.1. *Located reductions* $(u_0, v_0 \to v_0', w_0)$ *and* $(u_1, v_1 \to v_1', w_1)$ *are* compatible *if* $v_0 = v_1$ *and* $v_0' = v_1'$.

So now the compatible reductions are grouped together and the groups will be processed individually. Each group will be transformed into a single rewriting meta-instruction as follows. Let $\{(u_i, v \to v', w_i); i \in I\}$ be a group of compatible reductions. The inferred meta-instruction will be $(\mathsf{ZR}(\{u_i; i \in I\}), v \to v', \mathsf{ZR}(\{w_i; i \in I\}))$.

Then the only accepting meta-instruction is obtained by applying the ZR algorithm to the set of all simple words obtained from the input, let it be B. Thus, the meta-instruction is $(\mathsf{ZR}(B), \mathrm{Accept})$.

Finally, the set of meta-instructions of the output restarting automaton consists of all the created meta-instructions and the alphabets are taken from the input of the problem.

We will now present our Omega2 method[1] more formally. It is a method for solving the ABR inference from positive samples problem.

[1]The name Omega was used as an alternative name for the learning method presented in [35].

Algorithm 6.3.2. Omega2 method.
Input: a finite set of FPABR samples S, an input alphabet Σ, and a working alphabet Γ.
Output: a S-ZR-RRWW-automaton consistent with S.

1. If S is empty, then stop and return $M = (\Sigma, \Gamma, \emptyset)$.
2. Transform the set S into the set of located reductions (using the first-opportunity rewritings) $R = \{(u_i, v_i \to v'_i, w_i); i \in I\}$.
3. Transform each maximal group of compatible reductions from R into the rewriting meta-instruction as described above to obtain rewriting meta-instructions $RW = \{(E_{\ell,j}, x_j \to x'_j, E_{\mathrm{r},j}); j \in J\}$ (where J is the set of groups of compatible reductions).
4. Transform the set B of simple words from samples S into the accepting meta-instruction $accept = (\mathsf{ZR}(B), \mathrm{Accept})$.
5. Return the restarting automaton $M = (\Sigma, \Gamma, RW \cup \{accept\})$.

It is easy to see that the following holds.

Theorem 6.3.3. *Given a finite set of FPABR samples S,* Omega2 *returns a* S-ZR-RRWW*-automaton consistent with S.*

This is our basic method for inferring S-ZR-RRWW-automata from FPABR samples. It was further improved as presented in the next section.

6.4 Omega* Method

Though interesting results comparable to the results of other researchers can be achieved using the approach presented in Section 6.3 (see Subsection 7.3.1 for details), the results obtained in benchmarks using random targets were rather poor as presented in Subsection 7.3.2. Therefore, we decided to extend the class of considered models by allowing a richer class of languages in place of the zero-reversible languages. Note that we can easily turn our Omega2 method into a general schema into which different algorithms for inferring languages from samples can be supplied (note that the consistency of the resulting restarting automaton with the given input samples will be assured only when the supplied learning algorithm returns results consistent with its input):

Algorithm 6.4.1. A general schema for inferring S-RRWW-automata from FPABR samples.
Input: a finite set of FPABR samples S, an input alphabet Σ, a working alphabet Γ, and a learning algorithm F which for any set T of words returns a FSA $F(T)$ accepting all words from T.
Output: a S-RRWW-automaton consistent with S.

1. If S is empty, then stop and return $M = (\Sigma, \Gamma, \emptyset)$.
2. Transform the set S into the set of located reductions (using the first-opportunity rewritings) $R = \{(u_i, v_i \to v'_i, w_i); i \in I\}$.

3. Transform each maximal group of compatible reductions $\{(u_i, v \to v', w_i); i \in I'\} \subseteq R$ into the rewriting meta-instruction

$$(\mathrm{F}(\{u_i; i \in I'\}), v \to v', \mathrm{F}(\{w_i; i \in I'\}))$$

to obtain the set of rewriting meta-instructions RW.

4. Transform the set B of simple words from samples S into the accepting meta-instruction $accept = (\mathrm{F}(B), \mathrm{Accept})$.

5. Return the restarting automaton $M = (\Sigma, \Gamma, RW \cup \{accept\})$.

This way, it suffices to supply some suitable inference algorithm F (e.g. ZR) to obtain a new algorithm for learning single restarting automata from FPABR samples. We will exploit it in this section.

The idea of 0-reversibility can be generalized to obtain a hierarchy of classes of k-reversible languages [4]. Consequently, by considering richer classes, we increase the power of our model (as it will be proved later).

Definition 6.4.2 ([4]). *Let $k \geq 0$. Let $A = (Q, \Sigma, \delta, I, F)$ be a NFSA. Then*

- *$u \in \Sigma^k$ is a* k-follower of $q \in Q$ in A *if $\delta(q, u) \neq \emptyset$.*
- *$u \in \Sigma^k$ is a* k-leader of $q \in Q$ in A *if $\delta^{\mathrm{R}}(q, u^{\mathrm{R}}) \neq \emptyset$.*
- *A is* deterministic with lookahead k *if for any pair of states $p, q \in Q$ such that $p \neq q$, if $p, q \in I$ or $p, q \in \delta(r, a)$ for some $r \in Q, a \in \Sigma$, then there is no word that is a k-follower of both p and q.*
- *A is* k-reversible *if A is deterministic and A^{R} is deterministic with lookahead k.*
- *A language L is* k-reversible *if there is a k-reversible automaton A such that $L = \mathrm{L}(A)$.*
- *A language L is* reversible *if it is k-reversible for some k.*

Let A be a NFSA that is deterministic with lookahead k. Note that if A needs to perform a non-deterministic choice while reading an input word (either which initial state to start with or what state to enter next), it can resolve that situation by reading the following k symbols because there will always remain at most one way for any combination of k symbols. Also note that the definition for the case of $k = 0$ coincides with the definitions for 0-reversible automata and languages presented earlier.

We present here some basic properties of k-reversible languages to give you an idea of what they look like:

Theorem 6.4.3 ([4]). *Let $k \geq 0$ and Σ be an alphabet. Let R_k be the class of k-reversible languages over Σ and let $R_* = \bigcup_{\ell \geq 0} R_\ell$. Then it holds:*

- *R_* contains all the finite languages over Σ,*
- *R_* does not contain all regular languages over Σ,*

- $R_k \subset R_{k+1}$,
- R_k *is closed under pairwise intersection,*
- R_k *is closed under reversal, i.e. for* $L \in R_k$ *it holds* $L^{\mathrm{R}} \in R_k$,
- R_* *is not closed under pairwise union or complementation,*
- R_* *is not closed under concatenation, and*
- R_* *is not closed under Kleene closure.*

Below we present some sample languages, both reversible and not reversible ones:

- L($\oplus 0^*1 + \otimes(00)^*1$) is not a reversible language [4] (but note that it is a regular language over the alphabet $\{0, 1, \oplus, \otimes\}$).
- $\{0^\ell; \ell > k\}$ is a $(k+1)$-reversible language but not a k-reversible language for a constant $k \geq 0$ [4].
- $\{0^{2n}; n \geq 0\}$ is a 0-reversible language. Note that this language is not strictly locally testable.
- L(0^*1^*) is a 1-reversible language [4].

There is an important result [42] stating that the class of strictly locally testable languages is *properly* included in the class of reversible languages. This is of great importance because it shows that considering reversible languages leads to a greater class of languages to be used in meta-instructions compared to the strictly locally testable languages used in [55] (the method presented in Section 6.1). Moreover, according to [72], the class of strictly k-testable languages is properly included in the class of $(k+1)$-reversible languages.

We summarize some of the above mentioned relations in Fig. 6.1. Note particularly the relation of reversible and strictly locally testable languages. The fact that the class of strictly locally testable languages contains all finite languages can be easily seen from the Definition 6.1.1 of strictly k-testable languages because words shorter than k are not required to meet any criteria. For a finite language L, it thus suffices to pick the value of k larger than the length of the longest word from L and then the triple $(\emptyset, \emptyset, \emptyset)$ represents the strictly k-testable language L. Thus, each finite language is also k-reversible for some $k \geq 0$.

Similarly to the case of 0-reversible languages, there is an algorithm that can be used to effectively infer k-reversible languages from positive samples [4]. We will introduce it below:

Algorithm 6.4.4. Algorithm k-RI.
Input: a nonempty finite set S of positive sample words and the reversibility level $k > 0$.
Output: a k-reversible finite state automaton A accepting all words from S.

Let $A_0 = (Q_0, \Sigma, \delta_0, I_0, F_0)$ be PTA(S).
Let π_0 be the trivial partition of Q_0.
Let $i = 0$.

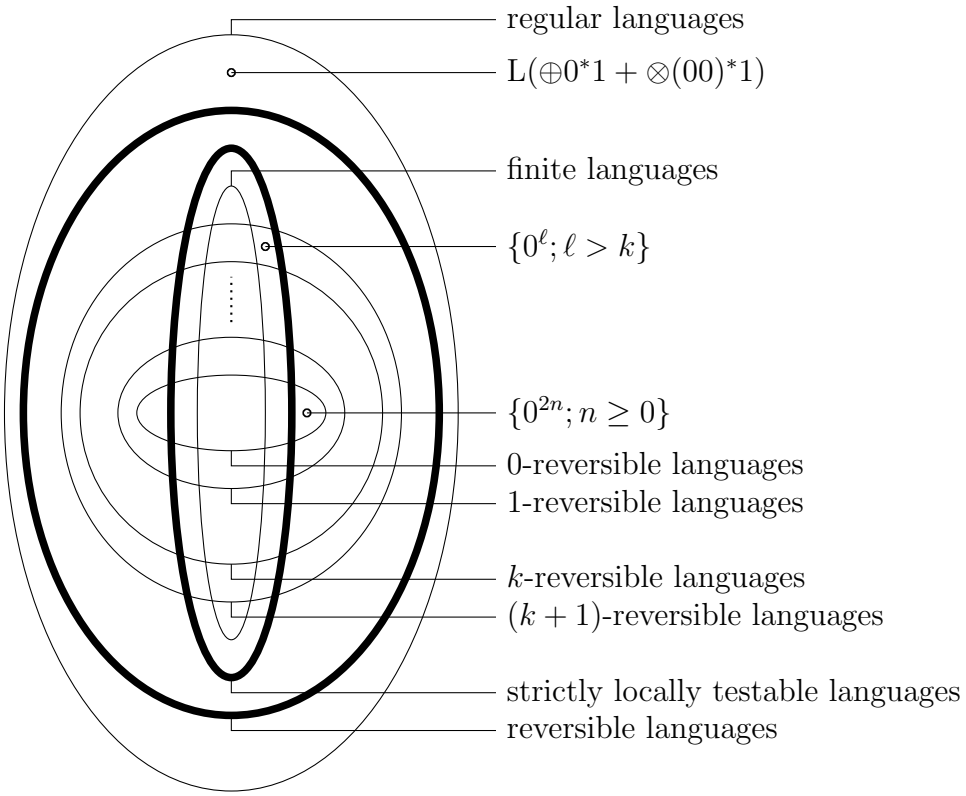


Figure 6.1: Relations regarding reversible languages.

while there are $B_1, B_2 \in \pi_i$ such that $B_1 \neq B_2$ and either
a) there are $b \in \Sigma, B_3 \in \pi_i$ such that B_1, B_2 are both b-successors of B_3 in A_0/π_i, or
b) B_1, B_2 have a common k-leader in A_0/π_i and either B_1 and B_2 are both accepting states of A_0/π_i or there are $B_3 \in \pi_i, b \in \Sigma$ such that B_3 is a b-successor of both B_1 and B_2 in A_0/π_i
do
begin
Merge B_1 with B_2 in π_i to obtain π_{i+1}.
Increase i by 1.
end
return a canonical acceptor for $\mathrm{L}(A_0/\pi_i)$.

The following theorem describes the output of the k-RI algorithm.

Theorem 6.4.5 ([4])**.** *Let S be a nonempty finite set of sample words, let $k > 0$ be the reversibility level, and let A be the acceptor returned by k-RI on S and k. Then $\mathrm{L}(A)$ is the smallest k-reversible language containing S.*

Having this algorithm, it is easy to modify our original approach to consider richer classes of languages to be used in contexts and in accepting meta-instructions — we call our new method Omega*. It is very similar to Omega2 — we just use the k-RI algorithm instead of the ZR algorithm. For example, each

group of compatible reductions will be transformed into one rewriting meta-instruction as follows: Let $\{(u_i, v \to v', w_i); i \in I\}$ be a group of compatible reductions. The inferred meta-instruction will be $(k\text{-RI}(\{u_i; i \in I\}), v \to v', k\text{-RI}(\{w_i; i \in I\}))$ for some value of k specified as input to Omega*.

This approach leads to a generalization of single zero-reversible restarting automata to single k-reversible restarting automata defined below:

Definition 6.4.6. *A* rewriting meta-instruction $(E_\ell, x \to y, E_\mathrm{r})$ *is called* k-reversible*, if both E_ℓ and E_r are k-reversible languages.*

An accepting meta-instruction (E, Accept) *is called* k-reversible*, if E is a k-reversible language.*

A restarting automaton *is called* k-reversible *(kR-RRWW-automaton), if all its meta-instructions are k-reversible.*

A restarting automaton *is called* single k-reversible *(S-kR-RRWW-automaton), if it is both single and k-reversible.*

A S-kR-RRWW-automaton not using auxiliary symbols is called a S-kR-RRW-automaton. A S-kR-RR-automaton is a S-kR-RRW-automaton such that for each rewriting meta-instruction $(E_\ell, u \to v, E_\mathrm{r})$, it holds that v can be obtained from u by deleting some symbols from u.

*The union of all classes of S-kR-RRWW-automata, S-kR-RRW-automata, and S-kR-RR-automata, for all $k \geq 0$, is denoted by S-*R-RRWW, S-*R-RRW, and S-*R-RR, respectively.*

*For $X \in \{$S-*R-RRWW, S-*R-RRW, S-*R-RR, S-kR-RRWW, S-kR-RRW, S-kR-RR, kR-RRWW$\}$, we denote by $\mathcal{L}(X)$ the class of languages that can be accepted by an X-automaton.*

Note, that the definition of S-0R-RRWW-automata coincides with the definition of S-ZR-RRWW-automata. The same holds for other definitions for $k = 0$ as can be easily seen.

We already know that even our simplest model of S-ZR-RRWW-automata has enough power to accept all languages from GCSL. Below we show that S-kR-RRW-automata form a hierarchy with respect to the degree of reversibility k.

Theorem 6.4.7. *For all $k \geq 0$, it holds:*

- $\mathcal{L}$(S-kR-RRW) $\subset \mathcal{L}$(S-$(k+1)$R-RRW)*, and*
- $\mathcal{L}$(S-kR-RR) $\subset \mathcal{L}$(S-$(k+1)$R-RR)*.*

Proof. For each $k \geq 0$, the language $L_{\neg k} = \{0^\ell; \ell > k\}$ is known to be $(k+1)$-reversible but not k-reversible [4]. The following sample language was designed having this fact in mind. Let us consider the language $L = L_1 \cup L_2$, where

- $L_1 = \{0^r 1^s 0^t; r > k, s > 0, \text{ and } r = t\}$, and
- $L_2 = \{0^r 1^s 0^t; 0 \leq r \leq k, s > k, \text{ and } t > s\}$.

We will show that $L \in \mathcal{L}$(S-$(k+1)$R-RR $\setminus \mathcal{L}$(S-kR-RRW). We will start by proving that $L \in \mathcal{L}$(S-$(k+1)$R-RR). We define $B = (\{0,1\}, \{0,1\}, I_B)$, where I_B contains the following meta-instructions:

- $i_1 = (\{0^i; i > k\}, 1 \to \lambda, \{1\} \cdot \{0,1\}^*)$ (note the language $L_{\neg k}$ used as the left context of this meta-instruction),
- $i_2 = (\{0^i; i > k\}, 010 \to 1, \{0\}^*)$ (note the language $L_{\neg k}$ again),
- $i_3 = (\{0^m 1^i; 0 \leq m \leq k, i > k\}, 10 \to \lambda, \{0\}^*)$,
- $i_4 = (\{0^m 1^{k+1} 0^{k+2}; 0 \leq m \leq k\}, 0 \to \lambda, \{0\}^*)$, and
- $i_5 = (\{0^{k+1}10^{k+1}\} \cup \{0^m 1^{k+1} 0^{k+2}; 0 \leq m \leq k\}, \text{Accept})$.

Claim. *It holds* $\mathrm{L}(B) = L$.

Proof. At first, we will prove $L \subseteq \mathrm{L}(B)$. Let $0^r 1^s 0^r \in L_1$ for some $r > k$ and $s > 0$. Then:

- If $s > 1$, then this number will be lowered using i_1 and a shorter word from L will be obtained.
- If $s = 1$ and $r > k + 1$, then the number of 0's will be lowered on both the left and the right end of the word using i_2 and a shorter word from L is thus obtained.
- If $s = 1$ and $r = k + 1$, then the current word will be accepted using i_5.

Further, let $0^r 1^s 0^t \in L_2$ for some $0 \leq r \leq k, s > k,$ and $s < t$. Then:

- If $s > k + 1$, the word is shortened by i_3 and the result still belongs to L.
- If $s = k + 1$ and $t > k + 2$, the number of trailing 0's is lowered by i_4 and the obtained word is still in L.
- If $s = k + 1$ and $t = k + 2$, the current word is accepted by i_5.

Thus, each word from L is accepted by B. Now we will show that $\mathrm{L}(B) \subseteq L$. Obviously, the words accepted by i_5 belong to L. It will suffice to prove that no word out of L can be directly reduced to a word from L. We will revert the way meta-instructions work, and rather than u can be reduced to v we will say v can be expanded to u. So our goal is to show that no word from L can be expanded to a word out of L. We will consider particular rewriting meta-instructions:

- In the case of i_1, we have the word $0^i 1u$ for some $i > k$ and $u \in \{0,1\}^*$. Here i_1 inserts the new symbol 1 right after the prefix 0^i and thus the new word is also from L.
- In the case of i_2, we have the word $0^i 10^j$ for $i > k$ and $j \geq 0$. Because i_2 both prepends and appends the symbol 1 with the symbol 0, the resulting word belongs to L.
- In the case of i_3, we have the word $0^m 1^i 0^j$ for some $m \leq k, i > k$, and $j \geq 0$. Here i_3 inserts 10 right after the end of the subword 1^i. Therefore, the word belongs to L as well.

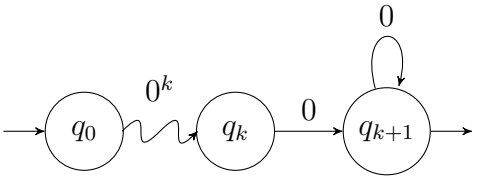


Figure 6.2: A $(k+1)$-reversible FSA accepting $\{0^i; i > k\}$.

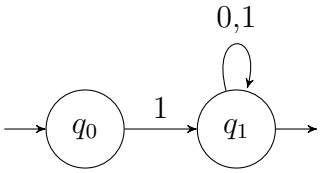


Figure 6.3: A 1-reversible FSA accepting $\{1\} \cdot \{0,1\}^*$.

- In the case of i_4, we have the word $0^m 1^{k+1} 0^{k+2} 0^j$ for some $m \leq k$ and $j \geq 0$. Because i_4 inserts another zero into the trailing sequence of 0's, the new word is from L.

In total, we start with a word from L (accepted using i_5) and only a stepwise expansion to another words from L is possible. So we know $\mathrm{L}(B) \subseteq L$ and thus B accepts L.

□

It remains to prove that B is the right type of a restarting automaton.

Claim. *The automaton B is a* S-$(k+1)$R-RR-*automaton.*

Proof. Obviously, B is an RR-automaton and all its meta-instructions are single. We need to prove that all languages used in meta-instructions are $(k+1)$-reversible:

- $\{0^i; i > k\}$ is $(k+1)$-reversible according to [4] (see Fig. 6.2 for a corresponding FSA).
- $\{1\} \cdot \{0,1\}^*$ is 1-reversible (see Fig. 6.3 for a corresponding 1-reversible FSA) and thus also $(k+1)$-reversible according to Theorem 6.4.3.
- $\{0\}^*$ is 0-reversible (the one-state FSA accepting this language is obviously 0-reversible) and thus also $(k+1)$-reversible.
- $\{0^m 1^i; 0 \leq m \leq k, i > k\}$ is $(k+1)$-reversible (see Fig. 6.4).
- $\{0^m 1^{k+1} 0^{k+2}; 0 \leq m \leq k\}$ is $(k+1)$-reversible (see Fig. 6.5).
- $\{0^{k+1} 1 0^{k+1}\} \cup \{0^m 1^{k+1} 0^{k+2}; 0 \leq m \leq k\}$ is just an extension of the previous language and it remains $(k+1)$-reversible as shown in Fig. 6.6.

□

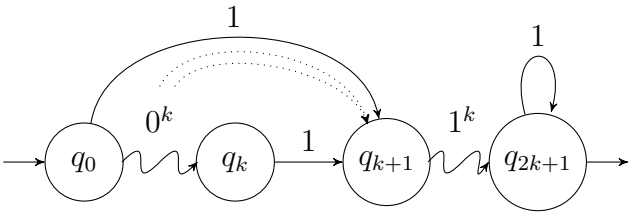


Figure 6.4: A $(k+1)$-reversible FSA accepting $\{0^m 1^i; 0 \leq m \leq k, i > k\}$. Note the dotted arrows denoting 1-transitions from all states present on the path from q_0 to q_k.

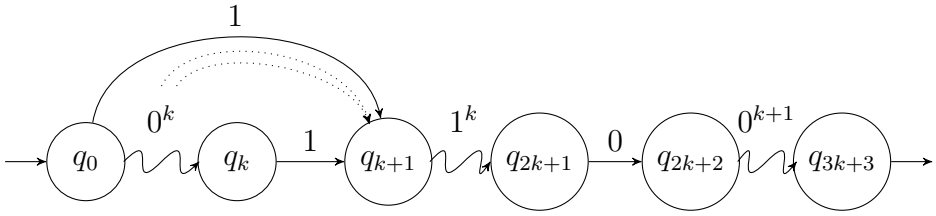


Figure 6.5: A $(k+1)$-reversible FSA accepting $\{0^m 1^{k+1} 0^{k+2}; 0 \leq m \leq k\}$. Note the dotted arrows denoting 1-transitions from all states present on the path from q_0 to q_k.

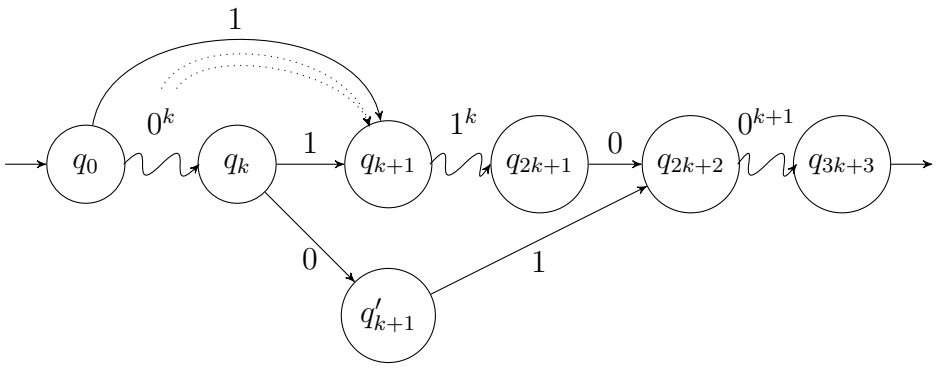


Figure 6.6: A $(k+1)$-reversible FSA accepting $\{0^{k+1}10^{k+1}\} \cup \{0^m 1^{k+1} 0^{k+2}; 0 \leq m \leq k\}$. Note the dotted arrows denoting 1-transitions from all states present on the path from q_0 to q_k.

It remains to prove that $L \notin \mathcal{L}(\mathsf{S}\text{-}k\mathsf{R}\text{-}\mathsf{RRW})$. For a contradiction, let us suppose there is a $\mathsf{S}\text{-}k\mathsf{R}\text{-}\mathsf{RRW}$-automaton $A = (\{0,1\}, \{0,1\}, I)$ such that $\mathrm{L}(A) = L$.

Our automaton B used the meta-instruction i_1 to shorten the segment of 1's of words from L_1 in order to accept them. The idea of proving that A can not accept L is to show that a similar reduction would lead to violating the error preserving property because A is k-reversible (A therefore can not use the language $L_{\neg k}$). We will prove that A reduces $0^k1^n0^n \notin L$ (for suitable $n > k$) to $0^k1^{n'}0^n \in L$ (where $k < n' < n$). We will therefore analyze how A works on the words $w_n = 0^n1^n0^n$ for $n > k$ from L. We will proceed in the following steps:

(P1) We will prove that in order to accept w_n (for sufficiently large values n), the automaton A first shortens the segment of 1's.

(P2) The meta-instructions used in reductions mentioned in (P1) can be applied to several words. One of the conditions is specified by the left contexts of those meta-instructions. The left contexts of those meta-instructions are k-reversible languages. Let the union of those left contexts be LC. We will show how to cover $\mathrm{RQ}(\mathrm{LC}, \mathrm{L}(1^*)) \cap \mathrm{L}(0^*)$ (i.e. the initial segments of 0's of words that can be reduced using the considered meta-instructions) with a finite set of k-reversible languages.

(P3) We will prove that 0^k is covered by the finite set of k-reversible languages from step (P2).

(P4) We will use this 0^k to obtain the above mentioned reduction that violates the error preserving property.

Let us start with (P1) in the form of the following claim.

Claim. *For sufficiently large values of n, the words w_n must be reduced at first in order to be accepted by A. The only way how to reduce those words in an accepting computation is to decrease the number of symbols* 1.

Proof. By using the standard pumping lemma for regular languages (Theorem 1.1.25) on w_n for sufficiently large n, it can be easily shown that there are only finitely many values $n' > k$ such that $w_{n'}$ is accepted by the accepting meta-instruction of A.

Let k' be the size of the window of A. Let us have an accepting computation on w_n such that it consists of at least one reduction and $n > k + k'$. Thus, it holds $w_n \vdash_A w'$ for some $w' \in \{0,1\}^*$. If $w' \in L_2$, then A surely shortened the initial segment of 0's. However, at most k' 0's can be removed at once by A. Therefore, A has to perform a rewriting using a replacement word starting with 1 near the left end of w, but this obviously yields a word $w' \notin L_2$. The only other way is that $w' \in L_1$. As $n > k + k'$, it is not possible to rewrite in both segments of 0's. If exactly one segment of 0's is influenced by the rewriting, then due to the fact that A is a single automaton, we obtain that the segment of 0's is shortened. This would lead to $w' \notin L_1$. Therefore, no segment of 0's can be rewritten in the first step. So the rewritten word consists of 1's only. Again, as we have a single automaton, it is not possible for the replacement word to start or to end with the symbol 1. When the symbol 0 is present in the replacement

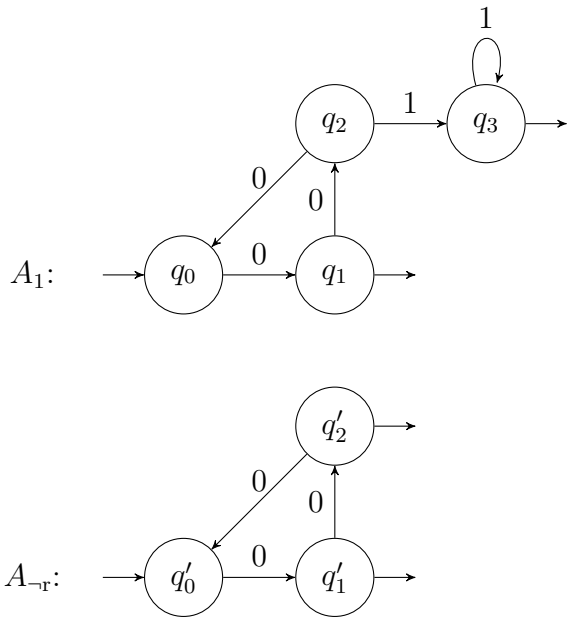


Figure 6.7: An example of a 1-reversible FSA A_1 accepting language $\mathrm{L}((0(000)^*)+(00(000)^*11^*))$ and a FSA $A_{\neg r}$ accepting the language $\mathrm{RQ}(\mathrm{L}(A_1), \mathrm{L}(1^*)) \cap \mathrm{L}(0^*)$ that is not a reversible language.

word, then we obtain $w' \notin L_1$. Finally, we see that some 1's were replaced with λ.

□

Let n_{mla} be the maximal length of a word accepted by A directly and $n_{\mathrm{reduce}} = k + k' + n_{\mathrm{mla}}$ (where k' is the size of the window of A). Thus, for all $n > n_{\mathrm{reduce}}$, the word w_n can be surely reduced by A by shortening the segment of 1's (by using the empty replacement word). This concludes the step (P1).

Let us proceed with (P2). We would like to show that A performs a reduction that violates the error preserving property. Let I' be the set of all meta-instructions of A that reduce some w_n ($n > n_{\mathrm{reduce}}$) to a word from L. We would like to collect all words 0^n such that at least one of the considered meta-instructions from I' reduces a word having a prefix 0^n1. If LC is the union of the left contexts of the meta-instructions I', then the language we are looking for is $\mathrm{RQ}(\mathrm{LC}, \mathrm{L}(1^*)) \cap \mathrm{L}(0^*)$. In order to show a contradiction, we need to cover this language with finitely many k-reversible languages.

Unfortunately, this language need not be k-reversible. Even when we consider a left context L_ℓ of some k-reversible meta-instruction individually (the left context would therefore be a k-reversible language), the language $\mathrm{RQ}(L_\ell, \mathrm{L}(1^*)) \cap \mathrm{L}(0^*)$ need not be k-reversible. See e.g. Fig. 6.7 — after transforming the language accepted by a 1-reversible FSA A_1, we obtain the language that is not reversible at all (this can be easily seen as the two accepting states q'_1 and q'_2 always share a common n-predecessor of any given length n). Therefore, we slightly modify the rewriting meta-instructions from I'. The automaton will not remain single but we do not need this property for the rest of this proof.

Let $(L_\ell, w \to w', L_r) \in I'$. Instead of this meta-instruction, we will consider meta-instructions defined below as (T1), (T2), and (T3). Let $D = (Q, \{0, 1\}, \delta, q_0,$

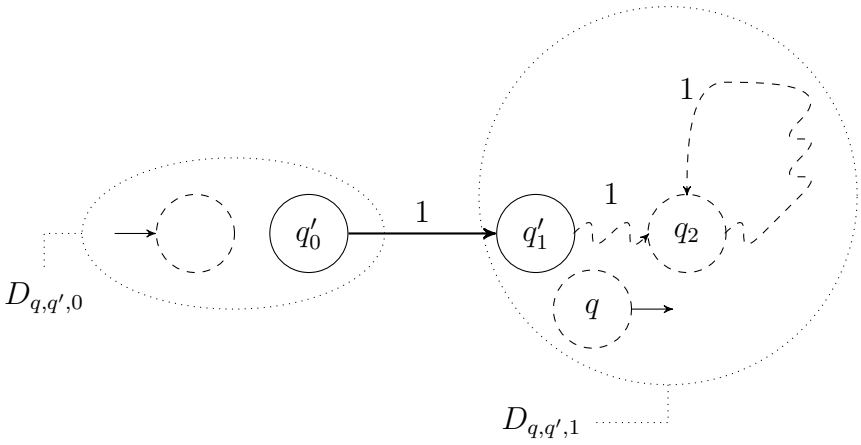


Figure 6.8: An illustration of $D_{q,q'}$.

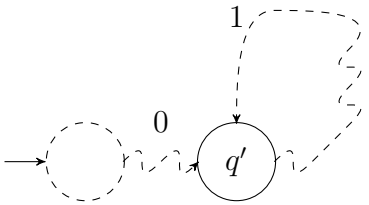


Figure 6.9: An illustration of $D'_{q,q'}$.

F) be a k-reversible FSA such that $\mathrm{L}(D) = L_\ell$. For each accepting state q of D, we create a copy of D called $D_q = (Q, \{0,1\}, \delta, q_0, \{q\})$ where only the state q is accepting.

(T1) If L_ℓ contains at least one word containing symbol 1, we perform the following. For each 1-transition from a state q' of D_q such that $\delta^*(q', 1^c) \neq q'$ for all $c > 0$ and q' is a state reachable from the initial state by 0-transitions only, we create $D_{q,q'}$ composed of two parts called $D_{q,q',0}$ and $D_{q,q',1}$. $D_{q,q',0}$ is a copy of D_q with no accepting state and with 1-transitions removed. On the other hand, $D_{q,q',1}$ is a copy of D_q with no initial state and with 0-transitions removed. Let the states of $D_{q,q',0}$ and $D_{q,q',1}$ be called $Q_{q,q',0}$ and $Q_{q,q',1}$, respectively. The resulting FSA $D_{q,q'}$ is equal to $(Q_{q,q',0} \cup Q_{q,q',1}, \{0,1\}, \delta_{q,q'}, I_{q,q'}, F_{q,q'})$ where $\delta_{q,q'}$ preserves the transitions of $D_{q,q',0}$ and $D_{q,q',1}$, and, in addition, there is the transition $\delta_{q,q'}(q'_0, 1) = q'_1$ where q'_0 is the copy of q' present in $Q_{q,q',0}$ and q'_1 is the copy of $\delta(q', 1)$ in $Q_{q,q',1}$. You can see an illustration of $D_{q,q'}$ in Fig. 6.8. The new meta-instructions are obtained as $(\mathrm{L}(D_{q,q'}), w \to w', L_\mathrm{r})$ for all possible values of q and q'.

(T2) For each 1-transition of D_q such that $\delta(q', 1^d) = q'$ for some $d > 0$ (let d be the smallest possible) where q' is reachable from the initial state by 0-transitions only and $\delta(q', 1^e) = q$ for some $e \geq 0$ (e smallest possible), we create $D'_{q,q'}$ obtained from D_q by removing all the 1-transitions and adding a single cycle of 1-transitions starting and ending at q' and having the length d, i.e. we add states $q'_1, \ldots, q'_{d-1}$ and the transitions will be $\delta(q', 1) =$

$q'_1, \delta(q'_i, 1) = \delta(q'_{i+1})$ for $i < d-1$, and $\delta(q'_{d-1}, 1) = q'$. If $q \neq q'$, then we put the set of accepting states equal to $\{q'_e\}$. You can see an illustration of $D'_{q,q'}$ in Fig. 6.9. The new meta-instruction will then be $(\mathrm{L}(D'_{q,q'}), w \to w', L_\mathrm{r})$.

(T3) For each D_q created above such that q is reachable by 0-transitions only, the new meta-instruction will be $(\mathrm{L}(D_q) \cap \mathrm{L}(0^*), w \to w', L_\mathrm{r})$.

It is obvious that the following claim holds.

Claim. *All words from L reduced by the original meta-instruction $(L_\ell, w \to w', L_\mathrm{r})$ can be reduced by some of the newly created ones the same way. On the other hand, our new meta-instructions do not perform any reduction not performed by the original meta-instructions.*

Proof. We know that $w \in \{1\}^*$. Therefore, if $w_\ell w w_\mathrm{r} \in L$ (for some $w_\ell, w_\mathrm{r} \in \{0,1\}^*$) is reduced by the original meta-instruction, then $w_\ell \in L_\ell$ satisfies $w_\ell \in \mathrm{L}(0^*1^*)$. If w_ℓ contains at least one symbol 1, then the reduction is performed by the meta-instruction defined in (T1) or (T2) (according to the computation of the FSA representing L_ℓ). Otherwise, it is performed by the meta-instruction defined in (T3).

On the other hand, the new meta-instructions were obviously defined in a way that they perform only a subset of reductions performed by the original ones.

□

The performed modification ensures a very useful feature of the newly created meta-instructions (we will denote this feature with EXCH; it will be used later in this proof) as stated in the following claim.

Claim. *Let $(L_D, w \to w', L_\mathrm{r})$ be one of the meta-instructions created in steps (T1), (T2), or (T3). Whenever $0^i1^j, 0^{i'}1^{j'} \in L_D$ for $i, i', j, j' \geq 0$, it holds $0^i1^{j'} \in L_D$.*

Proof. This claim holds because the FSA representing the left context contains at most one 1-transition that can be used when reading the first occurrence of 1 (if there is any) in both 0^i1^j and $0^{i'}1^{j'}$.

□

As we already mentioned, the automaton is no more single, but this does not influence our proof. The important thing is that it remains k-reversible as we will prove now.

Claim. *After the above performed transformation, the automaton A remains k-reversible.*

Proof. It suffices to prove that the left contexts of the newly created meta-instructions are k-reversible. Let us analyze the case of $D_{q,q'}$ at first. Is it k-reversible? It is obvious that $D_{q,q'}$ is deterministic and that it has only one accepting state. Let us consider $D^\mathrm{R}_{q,q'}$. It has only one initial state. Any non-determinism occurring in $D^\mathrm{R}_{q,q',0}$ was present in the original FSA D and thus does not violate the k-reversibility. The only place where non-determinism can occur in $D^\mathrm{R}_{q,q',1}$ is in the state q_2 if there is a cycle (it is possible that $q_1 = q_2$). We know that q'_0 and q'_1 are not a copy of the same state of D_q (as this was a special case handled by $D'_{q,q'}$). In D there was no common k-predecessor of both

states from which q_2 is reachable by a 1-transition. Therefore, the path from q_1 to q_2 using 1-transitions only could not be longer than $k-1$ — otherwise, as it could be prolonged by another symbol 1 leading to the copy of q'_0 in D, we would have a common k-predecessor of both the states from which q_2 is reachable by a 1-transition and D would not be k-reversible. Now, as the length is therefore at most $k-1$, the prolongation by the 1-transition from q'_1 to q'_0 leads to the word $1^{k'}$ where $k' \leq k$ that can not be further prolonged by any 1-transition as $D_{q,q',0}$ contains no 1-transitions. Thus, $D_{q,q'}$ is k-reversible.

In the case of $k = 0$, the above idea could contain an error if the original FSA D contains e.g. a 1-transition from q' to q' for some $q' \in Q$. This would be transformed into $D_{q,q'}$ such that $D^{\mathrm{R}}_{q,q'}$ would not be deterministic and thus also not 0-reversible. Similar problems occur when the length of the cycle using symbol 1 is more than one. Therefore, another approach is needed in all those cases. That is why we introduced $D'_{q,q'}$. It is clear that this FSA is k-reversible as it contains exactly one accepting state, and we did not introduce any new k-predecessor to any state where non-determinism occurs in the reverse of $D'_{q,q'}$ compared to the corresponding states of D_q.

Also, the case of $\mathrm{L}(D_q) \cap \mathrm{L}(0^*)$ can be easily seen to be k-reversible (marking some states as non-accepting preserves k-reversibility from D as well as intersecting with a 0-reversible language).

□

Let $L_{\ell,I'}$ be the set of left contexts of meta-instructions from I' (the union of those languages was denoted LC in (P2)) after the just performed transformation.

We will now transform $L_{\ell,I'}$ into $L_{\ell,I',0}$ in the following way. For each $L_i \in L_{\ell,I'}$, we perform the following steps to obtain $\mathrm{RQ}(L_i, \mathrm{L}(1^*)) \cap \mathrm{L}(0^*)$ (let A_i be a k-reversible FSA accepting L_i):

1. We mark each state of A_i that has an outgoing 1-transition as accepting (the states that were accepting before remain accepting as well).
2. We remove all 1-transitions and then also the non-reachable states.
3. We denote the resulting FSA as A'_i.
4. We put $\mathrm{L}(A'_i)$ into $L_{\ell,I',0}$.

Obviously, for each w_n ($n > n_{\mathrm{reduce}}$), there is a member of $L_{\ell,I',0}$ containing 0^n. Note that for each $0^r \in \mathrm{L}(A'_i)$, there is some $s \geq 0$ such that $0^r 1^s \in \mathrm{L}(A_i) = L_i$.

It is very important to note that all members of $L_{\ell,I',0}$ are k-reversible (remember that the automaton A'_i contains exactly one accepting state and the transitions were present in the k-reversible FSA A_i as well).

Thus, $L_{\ell,I',0}$ covers the language $\{0^n; n > n_{\mathrm{reduce}}\}$ with finitely many k-reversible languages because each w_n with $n > n_{\mathrm{reduce}}$ can be reduced by a meta-instruction from I'. By $L_{\ell,I',0,\mathrm{inf}}$ we denote the set languages from $L_{\ell,I',0}$ that originate in rewriting meta-instructions that reduce infinitely many different words w_n. Note that the languages from $L_{\ell,I',0,\mathrm{inf}}$ are necessarily infinite. Similarly as above, $L_{\ell,I',0,\mathrm{inf}}$ covers $\{0^n; n > n_1\}$ for some n_1.

We will show that necessarily $L_{\ell,I',0,\mathrm{inf}}$ covers also the word 0^k (step (P3)).

Claim. *The word 0^k belongs to at least one language from $L_{\ell,I',0,\mathrm{inf}}$.*

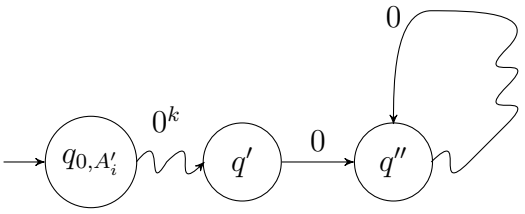


Figure 6.10: An illustration of A'_i where $\delta^*_{A'_i}(q_{0,A'_i}, 0^k)$ is not a member of the cycle.

Proof. Let us suppose the opposite holds. At first, let A'_i be a FSA accepting one of the languages from $L_{\ell,I',0,\inf}$.

Surely, $A'_i = (Q_{A'_i}, \{0\}, \delta_{A'_i}, \{q_{0,A'_i}\}, \{q_{A'_i}\})$ does not accept 0^k (we supposed that). Moreover, A'_i contains a cycle (because it accepts an infinite language). Because of the k-reversibility, $\delta^*_{A'_i}(q_{0,A'_i}, 0^k)$ is a member of that cycle (see Fig. 6.10 for an example of the opposite situation (there $q' = \delta^*_{A'_i}(q_{0,A'_i}, 0^k)$) — the transitions causing non-determinism in $\delta^{\mathrm{R}}_{A'_i}$ must occur no more than k symbols from q_{0,A'_i}). Let the length of the cycle be $r_{A'_i}$ symbols. Then even $0^{k+r_{A'_i}s}$ is not accepted for all $s > 0$.

For brevity, we denote individual values of $r_{A'_i}$ as $r_1, \ldots, r_z$ (let $z = |L_{\ell,I',0,\inf}|$). Thus, words $0^{k+r_1 s_1}, \ldots, 0^{k+r_z s_z}$ for all $s_1, \ldots, s_z \geq 0$ are not accepted by individual FSA's.

We would like to show that there are infinitely many words not covered by the union of all languages from $L_{\ell,I',0,\inf}$. Let $s_i = h \cdot \Pi_{j \neq i} r_j$ for $h > 0$. Then for all i, j such that $0 < i, j \leq z$, it holds $k + r_i s_i = k + r_j s_j$ and thus $0^{k+r_1 s_1}$ is not a member of any language from $L_{\ell,I',0,\inf}$. Therefore, we obtained infinitely many words not covered by $L_{\ell,I',0,\inf}$ (it suffices to consider infinitely many different values of h). This is a contradiction. Therefore, 0^k is a member of at least one language from $L_{\ell,I',0,\inf}$.

□

Now we conclude this proof with the step (P4).

Claim. *The automaton A performs a reduction that violates the error preserving property.*

Proof. Let us consider the word $w_n = 0^n 1^n 0^n \in L$ for some sufficiently large value of n that can be reduced using the rewriting meta-instruction with left context containing either 0^k or a word with prefix $0^k 1$ (and such that it could reduce infinitely many different words of this form). Let it be $(L_{\ell,\mathrm{p}}, 1^i \to \lambda, L_{\mathrm{r,p}})$ for some $i > 0$. So it reduces $0^n 1^{i_\ell} 1^i 1^{i_\mathrm{r}} 0^n \in L$ (where $i_\ell + i + i_\mathrm{r} = n$) into $0^n 1^{i_\ell + i_\mathrm{r}} 0^n$. Therefore, $0^n 1^{i_\ell} \in L_{\ell,\mathrm{p}}$ and thus also $0^k 1^{i_\ell} \in L_{\ell,\mathrm{p}}$ (thanks to the feature EXCH). Finally, we have that $0^k 1^{i_\ell} 1^i 1^{i_\mathrm{r}} 0^n = 0^k 1^n 0^n \notin L$ can be reduced to $0^k 1^{i_\ell} 1^{i_\mathrm{r}} 0^n \in L$ (we suppose n large enough to assure that $i_\ell + i_\mathrm{r} > k$).

□

This is a contradiction as obviously $\mathrm{L}(A) \neq L$. Thus, L can not be accepted by any S-kR-RRW-automaton.

□

It is an open question whether an analogical theorem holds in the case of automata using auxiliary symbols, i.e. in the case of S-kR-RRWW-automata. Thus, we have only the following trivial theorem. We include also the trivial case of kR-RRWW-automata.

Theorem 6.4.8. *For all $k \geq 0$ it holds:*

- $\mathcal{L}$(S-kR-RRWW) $\subseteq \mathcal{L}$(S-$(k+1)$R-RRWW), *and*
- $\mathcal{L}(k\text{R-RRWW}) \subseteq \mathcal{L}((k+1)\text{R-RRWW})$.

Proof. It is easy to see that by allowing more powerful languages in contexts of rewriting meta-instructions and in accepting meta-instructions, we can not limit the power of the models.

□

Another feature having an impact on the power of restarting automata is the kind of allowed rewriting meta-instructions that we analyze below.

Theorem 6.4.9. *It holds $\mathcal{L}$(S-kR-RR) $\subset \mathcal{L}$(S-kR-RRW) $\subset \mathcal{L}$(S-kR-RRWW) for all $k \geq 0$.*

Proof. It is obvious that $\mathcal{L}$(S-kR-RR) $\subseteq \mathcal{L}$(S-kR-RRW) $\subseteq \mathcal{L}$(S-kR-RRWW) holds as allowing more general rewriting meta-instructions can not decrease the power of the models. It thus remains to prove that the inclusions are proper. Let us prove $\mathcal{L}$(S-kR-RRW) $\setminus$ $\mathcal{L}$(S-kR-RR) $\neq \emptyset$. We will use the language $L = \{\oplus, \circ\circ\} \cdot \{0^n1^n; n \geq 0\} \cup \{\otimes, \circ\circ\} \cdot \{0^n1^m; m > 2n \geq 0\}$ that was shown not to be accepted by any RR-automaton in [37]. Below we prove that it can be accepted by a S-ZR-RRW-automaton $M = (\{\oplus, \otimes, \circ, 0, 1\}, \{\oplus, \otimes, \circ, 0, 1\}, I)$, where I contains the following meta-instructions:

(I1) $(\{\lambda\}, \circ\circ \to \oplus, \{0,1\}^*)$,

(I2) $(\{\lambda\}, \circ\circ \to \otimes, \{0,1\}^*)$,

(I3) $(\{\oplus\} \cdot \{0\}^*, 01 \to \lambda, \{1\}^*)$,

(I4) $(\{\otimes\} \cdot \{0\}^*, 011 \to \lambda, \{1\}^*)$,

(I5) $(\{\otimes 1\}, 1 \to \lambda, \{1\}^*)$, and

(I6) $(\{\oplus, \otimes 1\}, \text{Accept})$.

Let us take a word $w \in \{\oplus, \circ\circ\} \cdot \{0^n1^n; n \geq 0\}$. If w starts with $\circ\circ$, this prefix is replaced with $\oplus$ (using (I1)) at first. Then subwords 01 are iteratively removed using (I3), until the word consisting of single $\oplus$ is obtained and this word is then directly accepted by (I6). The case of a word $w \in \{\otimes, \circ\circ\} \cdot \{0^n1^m; m > 2n \geq 0\}$ is similar. If there is a prefix $\circ\circ$, it is replaced with $\otimes$ using (I2). Then subwords 011 are removed by (I4) while preserving the required form of the word. As the number of 1's is more than twice the number of 0's, there must be eventually some 1's left after all 0's have been removed. They will be removed by (I5) except the last one to obtain the word $\otimes 1$ that will be directly accepted by (I6).

On the other hand, let $w \notin L$ be the shortest word outside L accepted by M. It is surely not accepted directly. So it can be directly reduced to a word from L by a rewriting meta-instruction of M. Using (I1) or (I2) to replace $\circ\circ$ with either $\oplus$ or $\otimes$ obviously can not reduce w to any $w' \in L$. If (I3) was used to reduce w starting with $\oplus$, the removal of 01 in the middle of w preserves the difference of the number of 0's and the number of 1's and thus $w \in L$ as well. If (I4) was used to reduce w starting with $\otimes 0$, the removal of 011 in the middle of w again preserves the relation between the number of 0's and the number of 1's, so $w \in L$. The meta-instruction (I5) can only by applied onto a word from L, so we do not have to consider it now. So we conclude that $L = \mathrm{L}(M)$. Moreover, M is a S-ZR-RRW-automaton as can be checked directly according to Definition 6.2.8.

The relation $\mathcal{L}(\text{S-}k\text{R-RRW}) \subset \mathcal{L}(\text{S-}k\text{R-RRWW})$ follows easily from Theorem 6.2.9 and from the fact that to accept all context-free languages, auxiliary symbols are needed [37].

□

Also, thanks to the power of auxiliary symbols, each kR-RRWW-automaton can be transformed into a S-kR-RRWW-automaton accepting the same language.

Theorem 6.4.10. *For each* kR-RRWW*-automaton* M*, there is a* S-kR-RRWW*-automaton* M' *such that* $\mathrm{L}(M) = \mathrm{L}(M')$.

Proof. Let $M = (\Sigma, \Gamma, I)$ be a kR-RRWW-automaton. We will show how to transform M into a S-kR-RRWW-automaton M' such that $\mathrm{L}(M) = \mathrm{L}(M')$.

If $\mathrm{L}(M) = \emptyset$, then it is easy to create a S-kR-RRWW-automaton accepting the same language. Below we suppose that M accepts a non-empty language.

If a kR-RRWW-automaton is not a S-kR-RRWW-automaton, then at least one of the following conditions holds:

(i) it has at least two accepting meta-instructions,

(ii) it has at least two rewriting meta-instructions performing the same rewrite, or

(iii) it has at least one rewriting meta-instruction such that the rewritten and the replacement words share a common non-empty prefix or a common non-empty suffix.

We will deal with all those possible problems below.

Let us suppose that the condition (i) holds, i.e. let M have more than one accepting meta-instruction. Of course, it may not be possible to merge all accepting meta-instructions into one and to preserve k-reversibility at the same time. However, we can use the power of auxiliary symbols to process the words accepted directly as follows. Let L_{ad} be the set of all words accepted by M directly. It is surely a regular language. Thus, there is a DFSA $D = (Q, \Gamma, \delta, \{q_0\}, F)$ such that $L_{\mathrm{ad}} = \mathrm{L}(D)$. We define the following meta-instructions and we put them into I_{ad} (Q_q for $q \in Q$ are new auxiliary symbols):

- $(\lambda, ab \to Q_q, \Gamma^*)$ for each $a, b \in \Gamma$ and $q = \delta^*(q_0, ab)$,

- $(\lambda, Q_q a \to Q_{q'}, \Gamma^*)$ for each $q, q' \in Q, a \in \Gamma$, and $q' = \delta(q, a)$,
- for $\lambda \notin L_{\mathrm{ad}}$, we add $(\{Q_q; q \in F\} \cup (L_{\mathrm{ad}} \cap \Gamma), \mathrm{Accept})$, and
- for $\lambda \in L_{\mathrm{ad}}$, we add
  - $(\lambda, Q_q \to \lambda, \lambda)$ for all $q \in F$,
  - $(\lambda, x \to \lambda, \lambda)$ for all $x \in L_{\mathrm{ad}} \cap \Gamma$, and
  - $(\{\lambda\}, \mathrm{Accept})$.

The meta-instructions from I_{ad} can be surely used to accept exactly the words from L_{ad} and they can be safely used in place of the original accepting meta-instructions. Obviously, all meta-instructions from I_{ad} are zero-reversible and there is exactly one accepting meta-instruction. In what follows, we suppose that M has one accepting meta-instruction. Let it be $(L_{\mathrm{accept}}, \mathrm{Accept})$.

To deal with the condition (ii), we will need non-empty replacement words in nearly all rewriting meta-instructions. Then we will use new auxiliary symbols to make individual rewrites unique.

We will prepare this as follows. Let I_0 denote the set of all rewriting meta-instructions with a non-empty replacement word. Then, for each rewriting meta-instruction $(L_\ell, x \to \lambda, L_{\mathrm{r}}) \in I \setminus I_0$ (i.e. a meta-instruction with the empty replacement word) we distinguish the following cases:

(C1) If both L_ℓ and L_{r} contain only the empty word λ, we put this meta-instruction into I_λ.

(C2) If L_{r} contains some non-empty words, we collect the first symbols of all such words — let them form the set Δ. For each $a \in \Delta$, we insert the meta-instruction $(L_\ell, xa \to a, \mathrm{LQ}(L_{\mathrm{r}}, a))$ into I_0 (here again the right context remains k-reversible). You may notice that this meta-instruction satisfies the requirement mentioned in the condition (iii), but we will fix that later. If $\lambda \in L_{\mathrm{r}}$ and $L_\ell \neq \{\lambda\}$, we process $(L_\ell, x \to \lambda, \lambda)$ according to the step $(C3)$. If $\lambda \in L_{\mathrm{r}}$ and $L_\ell = \{\lambda\}$, we add $(\lambda, x \to \lambda, \lambda)$ into I_λ.

(C3) If L_{r} contains only λ but L_ℓ contains a non-empty word, we proceed similarly. We collect the last symbols of all such words — let them form the set Δ. For each $a \in \Delta$, we insert the meta-instruction $((\mathrm{LQ}(L_\ell^{\mathrm{R}}, a))^{\mathrm{R}}, ax \to a, L_{\mathrm{r}})$ into I_0. Again, we obtained a k-reversible meta-instruction — here we additionally used the fact that the class of k-reversible languages is closed under reversal (Theorem 6.4.3). Again, the problem with the condition (iii) will be fixed later. If $\lambda \in L_\ell$, we add $(\lambda, x \to \lambda, \lambda)$ into I_λ.

Obviously, the rewriting meta-instructions from $I_0 \cup I_\lambda$ perform the same (sliding) reductions as those from I. Note that no rewriting meta-instruction from I_0 uses the empty replacement word. Moreover, all those meta-instructions are k-reversible. We would like to use the meta-instructions from I_0 in M', but their presence could mean that M' still satisfies the condition (ii). Therefore, for each rewriting meta-instruction $r \in I_0$, we create three new alphabets $\Gamma_{r,0}$, $\Gamma_{r,1}$, and $\Gamma_{r,2}$, forming together an alphabet Γ_r (for $a \in \Gamma$ there will thus be three associated symbols $a_{r,i} \in \Gamma_r$ for $i \in \{0, 1, 2\}$ — we will further use a_r to

denote one of the symbols associated with a), we rewrite the replacement word in r with the associated symbols from $\Gamma_{r,0}$ in the obvious way, and we put the resulting meta-instruction into I_1. The purpose of $\Gamma_{r,1}$ and $\Gamma_{r,2}$ will be obvious later. Note that I_1 contains no pair of rewriting meta-instructions performing the same rewrite as the replacement words are unique — each one is composed of symbols from an alphabet specific to the particular meta-instruction.

Let $\Gamma_* = \Gamma \cup \bigcup_{r \in I_0} \Gamma_r$. Note that for each symbol $a \in \Gamma$, there can be therefore many associated symbols in Γ_* (there are three for each rewriting meta-instruction from I_0). We will say that a symbol $a \in \Gamma$ is *compatible* with itself and also with $a_{r,0}, a_{r,1}, a_{r,2} \in \Gamma_r$ for all $r \in I_0$. For a word w, we define a set of compatible words as $\mathrm{c}(w) = \{w' \in \Gamma_*^{|w|};$ for each i such that $0 \leq i < |w|$, it holds that w_i is compatible with $w'_i\}$. For a language $L_{\mathrm{orig}} \subseteq \Gamma^*$, we define $\mathrm{c}(L_{\mathrm{orig}}) = \bigcup_{w \in L_{\mathrm{orig}}} \mathrm{c}(w)$. For each rewriting meta-instruction $(L_\ell, x \to y, L_\mathrm{r}) \in I_1$ and each $x' \in \mathrm{c}(x)$, we add the meta-instruction $(\mathrm{c}(L_\ell), x' \to y, \mathrm{c}(L_\mathrm{r}))$ to I_2. The resulting meta-instruction is k-reversible as can be easily seen from the Definition 6.4.2. We will call the set of all meta-instructions created from the same meta-instruction in this step a *group*.

It is clear that groups can be distinguished by the replacement words because they remain the same as before and thus are group specific (remember that there were no duplicate replacement words in I_1). Also, no pair of meta-instructions from the same group share the same rewritten word.

Now no pair of rewriting meta-instructions from I_2 performs the same rewrite. However, the condition (iii) can still be a problem. We will inspect all rewriting meta-instructions as follows — let $r' = (L_\ell, x \to y, L_\mathrm{r}) \in I_2$. Note that $y \neq \lambda$ holds for all $r' \in I_2$. Let $x = awb$ where $a \in \Gamma_{r,i}, b \in \Gamma_{r,j}$, and $w \in \Gamma_r^*$ for some $r \in I_0$, then we rewrite $y \in \Gamma_{r,0}^*$ using symbols from $\Gamma_{r,\ell}$ where $\ell \in \{0,1,2\} \setminus \{i,j\}$ is selected arbitrarily — here we replace each symbol $a_r \in \Gamma_{r,0}$ with $a'_r \in \Gamma_{r,\ell}$ and then we insert this rewriting meta-instruction into I_3. Note that the meta-instructions from I_3 do not satisfy the requirements of the condition (iii). Also note that the new meta-instructions are still k-reversible. As the new replacement words remain unique for each group, no additional action is needed to avoid satisfying the condition (ii).

Remember the meta-instructions from I_λ. As we introduced new symbols in the previous steps, we need to adapt meta-instructions from I_λ to this new situation. Therefore, for each $(\lambda, w \to \lambda, \lambda) \in I_\lambda$, we put $(\lambda, w' \to \lambda, \lambda)$ into I'_λ for each $w' \in \mathrm{c}(w)$.

Finally, let $M' = (\Sigma, \Gamma_*, I_3 \cup I'_\lambda \cup \{(\mathrm{c}(L_{\mathrm{accept}}), \mathrm{Accept})\})$.

Note that the computation of M' resembles the computation of M. Only some compatible symbols occur in the processed words in place of the ones used by M.

Now, the following holds:

- M' contains exactly one accepting meta-instruction.
- Rewriting meta-instructions originating from the same group are distinguished by the rewritten word.
- Rewriting meta-instructions originating from different groups are distinguished by the alphabet used by the replacement word.

- Rewriting meta-instructions having the empty replacement word have contexts containing exactly the empty word.
- There is no rewriting meta-instruction performing a rewrite $x \to y$ such that x and y share the same non-empty prefix or the same non-empty suffix (note that the replacement word is either empty or it is composed of symbols from an alphabet that does not contain the first and the last symbol of the rewritten word).
- M' is k-reversible.

Thus, we obtained a S-kR-RRWW-automaton M'.

It remains to prove that $\mathrm{L}(M) = \mathrm{L}(M')$. It is clear that words accepted by M directly are accepted by M' directly as well, and the same holds for all words compatible with words accepted by M directly. If M reduces u to v, then each word $u' \in \mathrm{c}(u)$ (i.e. compatible with u) can be reduced by M' to a word compatible with v. Thus, $\mathrm{L}(M) \subseteq \mathrm{L}(M')$. We will now prove that $\mathrm{L}(M') \subseteq \mathrm{L}(M)$. Let $\mathrm{c}^{-1}(w)$ for $w \in \Gamma_*^*$ denote the unique word $w' \in \Gamma^*$ such that $w \in \mathrm{c}(w')$. If $w \in \mathrm{L}_\mathrm{c}(M')$ is accepted by M' directly, then surely $\mathrm{c}^{-1}(w) \in \mathrm{L}(M)$. If M' reduces $u \in \mathrm{L}_\mathrm{c}(M')$ to λ using a meta-instruction from I'_λ, then M reduces $\mathrm{c}^{-1}(u)$ to λ. If M' reduces $u \in \mathrm{L}_\mathrm{c}(M')$ to v using a meta-instruction from I_3, then M reduces $\mathrm{c}^{-1}(u)$ to $\mathrm{c}^{-1}(v)$. Overall, we have $\mathrm{L}(M') \subseteq \mathrm{L}(M)$.

□

Below we will show that there is a language that is accepted by a S-kR-RRW-automaton but not by any SLT-R-automaton not using auxiliary symbols.

Theorem 6.4.11. *Let $L = L_1 \cup L_2$, where*

- $L_1 = \{0^{2r}1^s0^{2r}1^t \mid r, s, t > 0\}$, *and*
- $L_2 = \{0^{2r+1}1^s0^t1^{s'}; r, s, s', t > 0$ *and* $s \neq s'\}$.

It holds:

- *There is no* SLT-R*-automaton using no auxiliary symbols that accepts L.*
- *There is a* S-kR-RR*-automaton accepting L.*

Proof. At first, we will show that L can not be accepted by any SLT-R-automaton. For a contradiction, let us suppose that there is a k'-SLT-R-automaton M such that $\mathrm{L}(M) = L$. Let us consider the word $w = 0^r1^s0^r1^s \in L_1$ for $r = 4k'$ and $s = 3k'$. Let us analyze the accepting computation on this word.

Below we will often use a very simple idea: Let L' be a strictly k'-testable language over an alphabet Γ. Let $u \in L'$ and let $v \in \Gamma^*$. If $\mathrm{P}_{k'}(u) = \mathrm{P}_{k'}(v), \mathrm{S}_{k'}(u) = \mathrm{S}_{k'}(v)$, and $\mathrm{I}_{k'}(u) = \mathrm{I}_{k'}(v)$, then $v \in L'$.

The accepting computation on w can start in any of the following ways:

- If w is directly accepted by M, then the word $0w$ is directly accepted as well and this is a contradiction.
- If w is directly reduced to some $w' \in L$, we distinguish two cases:

– To reduce w into a shorter word $w' \in L_1$, the only possible way is to lower the number of 1's in exactly one of the segments of 1's. If M lowers the number of 1's in one of the segments of 1's in w, then the same can be performed on the word $w'' = 0w \notin L$. However, this would reduce $w'' \notin L$ into $0w' \in L$. This is a contradiction.

– To reduce w into a shorter word $w' \in L_2$, we need to increase or decrease the number of 0's in the initial segment of 0's by an odd number and also to change the number of 1's in the first segment of 1's. If M reduces w this way, then surely the rewritten subword has the form 0^i1^j where $i, j \geq 0$. Let us consider the word $w'' = 00w \notin L$. Again, this can be reduced analogically into $00w' \in L$. This is a contradiction.

Overall, the word w can not be accepted by M. Thus, there is no SLT-R-automaton not using auxiliary symbols that accepts L. It remains to prove that L can be accepted by some S-kR-RRW-automaton. Let $B = (\{0,1\}, \{0,1\}, I)$ where I contains the following meta-instructions:

(I1) $i_1 = (\{0^{2r}; r \geq 0\}, 1 \to \lambda, \{1\} \cdot \{0,1\}^*)$,

(I2) $i_2 = (\{0^{2r}; r \geq 0\}, 00100 \to 1, \{0\} \cdot \{0,1\}^*)$,

(I3) $i_3 = (\{0^{2r+1}1^s; r \geq 0, s > 0\}, 0 \to \lambda, \{0\} \cdot \{0,1\}^*)$,

(I4) $i_4 = (\{0^{2r+1}1^s; r \geq 0, s > 0\}, 101 \to 0, \{1\} \cdot \{0,1\}^*)$, and

(I5) $i_5 = (\{001001^t; t > 0\} \cup \{0^{2r+1}101^t; r > 0, t \geq 2\} \cup \{0^{2r+1}1^s01; r > 0, s \geq 2\}, \text{Accept})$.

We will prove that $\mathrm{L}(B) = L$. We will start with the inclusion $L \subseteq \mathrm{L}(B)$.

- Let $w \in L_1$. At first, (I1) is used to reduce w into $0^{2r}10^{2r}1^t$ for some $r, t > 0$. This word is then reduced to 001001^t by (I2). Then (I5) is used to accept this word.

- Let $w \in L_2$. It is reduced into $0^{2r+1}1^s01^{s'}$ for some $r, s, s' > 0$ using (I3). Then (I4) is used to obtain the word $0^{2r+1}101^{s''}$ or $0^{2r+1}1^{s''}01$ for some $s'' > 1$ (depending on the relation between s and s'). The obtained word is then accepted by (I5).

Now we will prove that $\mathrm{L}(B) \subseteq L$.

- The words accepted by (I5) obviously belong to L.

- Let us revert the way meta-instructions work, and rather than u can be reduced to v we will say v can be expanded to u. By inspecting all rewriting meta-instructions of B, it is easy to see that no one expands a word from L into a word outside L. This yields the error preserving property.

In total, we have that $\mathrm{L}(B) = L$.

Obviously, B is a single restarting automaton. B is also an RR-automaton. It remains to prove that B is k-reversible for some k.

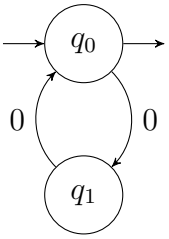


Figure 6.11: A 0-reversible FSA accepting $\{0^{2k}; k \geq 0\}$.

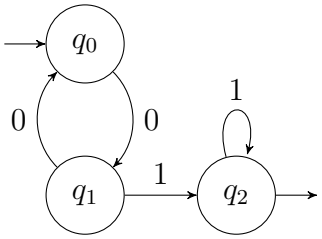


Figure 6.12: A 1-reversible FSA accepting $\{0^{2k+1}1^{\ell}; k \geq 0, \ell > 0\}$.

- i_1 is 1-reversible (see Fig. 6.11 and the proof of Theorem 6.4.7).
- i_2 is 1-reversible (similarly to i_1).
- i_3 is 1-reversible (see Fig. 6.12 and the proof of Theorem 6.4.7).
- i_4 is 1-reversible (similarly to i_3).
- i_5 is 3-reversible (see Fig. 6.13).

Finally, B is a S-3R-RR-automaton accepting L.

□

On the other hand, there is a language that is accepted by a SLT-R-automaton not using auxiliary symbols but not by any S-kR-RRW-automaton.

Theorem 6.4.12. *Let L be a language defined as follows:*

- $L_1 = \{a0^n c0^n c0^n a; a \in \{\oplus, \otimes\}, c \in \{\circ, \bullet\}, n \geq 0\}$,
- $L_2 = \{a0^n c0^{2n} c0^n b; a, b \in \{\oplus, \otimes\}, a \neq b, c \in \{\circ, \bullet\}, n \geq 0\}$,
- $L_3 = \{a0^{n-1} c0^{n-1} d0^n a; a \in \{\oplus, \otimes\}, c, d \in \{\circ, \bullet\}, c \neq d, n > 0\}$,
- $L_4 = \{a0^{n-1} c0^{2n-2} d0^n b; a, b \in \{\oplus, \otimes\}, a \neq b, c, d \in \{\circ, \bullet\}, c \neq d, n > 0\}$, *and*
- $L = L_1 \cup L_2 \cup L_3 \cup L_4$.

It holds:

- *There is no single* RRW*-automaton (and thus no* S-kR-RRW*-automaton) accepting L.*

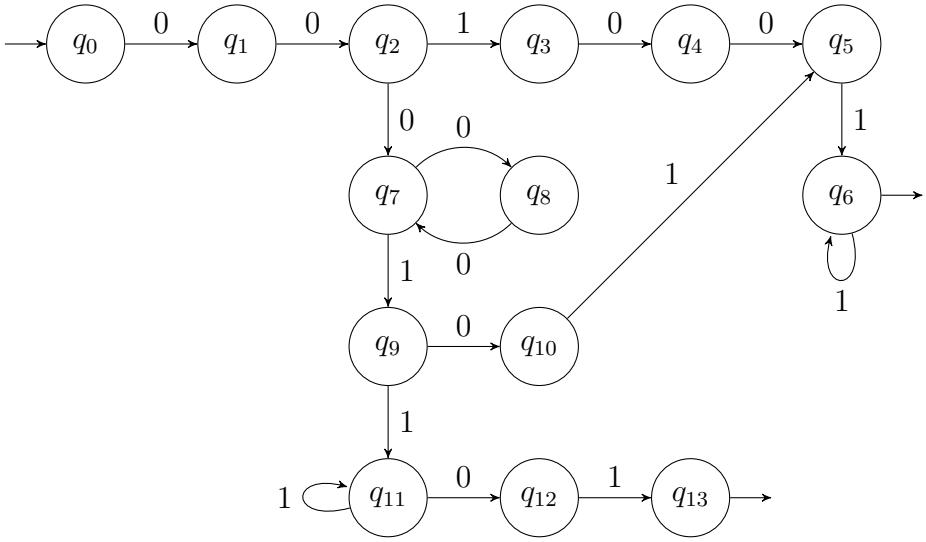


Figure 6.13: A 3-reversible FSA accepting the language $\{001001^t;\ t > 0\} \cup \{0^{2r+1}101^t;\ r > 0, t \geq 2\} \cup \{0^{2r+1}1^s01; r > 0, s \geq 2\}$.

- *There is a* SLT-R*-automaton using no auxiliary symbols that accepts* L*.*

Proof. We start with proving that no single RRW-automaton can accept L. Let us suppose the opposite holds, i.e. let M be a single RRW-automaton accepting L. Let us analyze how a reduction of a sufficiently long word from L can look like. We will consider particular words from L_2 — let us have a word $w = a0^n \circ 0^{2n} \circ 0^n b$, where $a, b \in \{\oplus, \otimes\}, a \neq b, n \geq 0$. For sufficiently large values of n, it is not possible to accept this word by an accepting meta-instruction of M (as can be easily proven using the pumping lemma (Theorem 1.1.25)). Thus, a reduction is needed. We suppose n greater than the size of the window of M. Below we analyze all possible ways how to reduce w:

- If the rewritten subword contains no delimiter from $\{\oplus, \otimes, \circ\}$, then the number of symbols present in exactly one of the segments consisting of 0's will be lowered (note that introducing a new delimiter would make the resulting word belong to the complement of L, so we can safely omit this case). Thus, the resulting word falls out of L.

- If the rewritten subword contains $\oplus$ or $\otimes$, then, as we have sufficiently long w, we can suppose that it does not contain both of them at once. Moreover, neither one of the symbols $\circ$ can be changed. The result of this reduction should fall into L_1 (the outer delimiters must be the same and the inner pair of the delimiters contains the original ones, i.e. $\circ$'s). However, as either the left segment or the right segment of 0's was shortened, the word does not belong to L_1.

- If the rewritten subword contains $\circ$, then we also need either $\circ$ or $\bullet$ in the replacement word to remain in L. As the rewritten subword must be longer than the replacement word and we have a single automaton, we know that at least one symbol 0 is removed. Moreover, the delimiters $\oplus$ and $\otimes$ are not changed by this reduction due to the limited size of the window of M.

Therefore, the word obtained by this reduction should belong to $L_2 \cup L_4$. Obviously, it is not possible to stay in L_2 after removing some 0's around either of the delimiters $\circ$. So there remains only one way, i.e. reducing into a word from L_4. Necessarily, the rewritten subword equals to $0^i \circ 0^j$ for some $i, j > 0$. Since we have a single automaton, we know that the replacement word equals to $\bullet$. It follows that $i = 1$ and $j = 2$.

Thus, we know that M contains a rewriting meta-instruction $rw = (L_\ell, 0 \circ 00 \to \bullet, L_\mathrm{r})$. We would like to proceed as follows. Let us have the word $w_{\overline{L}} = \oplus 0^{2n} 0 \circ 000^{2n} \circ 0^{2n+1} \oplus \notin L$. The idea is to show that this word can be reduced by rw to $w_L = \oplus 0^{2n} \bullet 0^{2n} \circ 0^{2n+1} \oplus \in L$. This will be a contradiction.

It is easy to see that $\oplus 0^{2n} \in L_\ell$ as the reductions $(\oplus 0^m, 0 \circ 00 \to \bullet, 0^{2m} \circ 0^{m+1} \otimes)$ must be performed (for sufficiently large m) by rw as shown above. It remains to prove that $0^{2n} \circ 0^{2n+1} \oplus \in L_\mathrm{r}$. At first, it may not be so obvious why this should hold. For a contradiction, let us suppose the opposite holds. Thanks to the closure properties of regular languages, we know that the complement of the language L_r, i.e. the language L_r^c, is regular as well. Let us suppose that for all $m > m_0$ (for some constant m_0), it holds $0^{2m} \circ 0^{2m+1} \oplus \in L_\mathrm{r}^\mathrm{c}$. It is clear that a DFSA A accepting L_r^c steps through some cycles while reading the segments of 0's in such words. We can pass such cycles several times and obtain new words belonging to L_r^c (similarly to the pumping lemma (Theorem 1.1.25)). Let us start with the word $0^{2n} \circ 0^{2n+1} \oplus \in L_\mathrm{r}^\mathrm{c}$ for sufficiently large n. We obtain words $0^{2n+k_1\ell_1} \circ 0^{2n+1+k_2\ell_2} \oplus \in L_\mathrm{r}^\mathrm{c}$ where ℓ_1, ℓ_2 are the lengths of corresponding cycles in A and $k_1, k_2 \geq 0$ are variables we can choose arbitrarily. We would like to find k_1 and k_2 such that $2(2n + k_2\ell_2) = 2n + k_1\ell_1$ (note that the value ℓ_2 may depend on n). Note that we can also choose $n > m_0$. Let $n = m_0 \cdot P! \cdot C$ where P is the number of states of A and $C > 0$ is an arbitrary number that assures that n is large enough to satisfy all restrictions on n introduced earlier in this proof. Now the values of ℓ_1, ℓ_2 are fixed. Let us compute:

$$\begin{aligned} 2(2n + k_2\ell_2) &= 2n + k_1\ell_1 \\ 4n + 2k_2\ell_2 &= 2n + k_1\ell_1 \\ 2n + 2k_2\ell_2 &= k_1\ell_1 \\ \frac{2n}{\ell_1} + \frac{2k_2}{\ell_1} \cdot \ell_2 &= k_1 \end{aligned}$$

We put $k_2 = \ell_1$ and thus k_1 is an integer. Thus, we have that

$$\begin{aligned} 0^{2n+k_1\ell_1} \circ 0^{2n+1+k_2\ell_2} \oplus &\in L_\mathrm{r}^\mathrm{c} \\ 0^{2n+2n+2\ell_1\ell_2} \circ 0^{2n+1+\ell_1\ell_2} \oplus &\in L_\mathrm{r}^\mathrm{c} \\ 0^{2(2n+\ell_1\ell_2)} \circ 0^{(2n+\ell_1\ell_2)+1} \oplus &\in L_\mathrm{r}^\mathrm{c} \\ 0^{2s} \circ 0^{s+1} \oplus \in L_\mathrm{r}^\mathrm{c} \text{ for } s &= 2n + \ell_1\ell_2 \end{aligned}$$

However, this means that the following word will not be reduced (we consider s as above) — and as n is large enough, it will not be accepted at all:

$$\otimes 0^{2n+\ell_1\ell_2} 0 \circ 000^{2(2n+\ell_1\ell_2)} \circ 0^{(2n+\ell_1\ell_2)+1} \oplus = \otimes 0^{s+1} \circ 0^{2(s+1)} \circ 0^{s+1} \oplus \in L_2 \subseteq L$$

Thus, we have a contradiction. Therefore, $0^{2n} \circ 0^{2n+1} \oplus \in L_\mathrm{r}$. Thus, M reduces the word $w_{\overline{L}} \notin L$ into the word $w_L \in L$. This is a contradiction. Therefore, the automaton M does not accept L.

It remains to prove that there is a SLT-R-automaton M' not using auxiliary symbols and accepting L. Let $\Sigma = \{0, \oplus, \otimes, \circ, \bullet\}$ and $M' = (\Sigma, \Sigma, I)$ where I contains the following meta-instructions (let $\Sigma_\circ = \Sigma \setminus \{\bullet\}, \Sigma_\bullet = \Sigma \setminus \{\circ\}$):

(I1) $(\oplus 0^*, 0 \circ 0 \to \bullet, \Sigma_\circ^* \oplus)$,

(I2) $(\otimes 0^*, 0 \circ 0 \to \bullet, \Sigma_\circ^* \otimes)$,

(I3) $(\oplus 0^*, 0 \bullet 0 \to \circ, \Sigma_\bullet^* \oplus)$,

(I4) $(\otimes 0^*, 0 \bullet 0 \to \circ, \Sigma_\bullet^* \otimes)$,

(I5) $(\oplus 0^*, 0 \circ 00 \to \bullet, \Sigma_\circ^* \otimes)$,

(I6) $(\otimes 0^*, 0 \circ 00 \to \bullet, \Sigma_\circ^* \oplus)$,

(I7) $(\oplus 0^*, 0 \bullet 00 \to \circ, \Sigma_\bullet^* \otimes)$,

(I8) $(\otimes 0^*, 0 \bullet 00 \to \circ, \Sigma_\bullet^* \oplus)$,

(I9) $(\Sigma_\circ^*, \bullet 0 \to \circ, 0^*(\oplus + \otimes))$,

(I10) $(\Sigma_\bullet^*, \circ 0 \to \bullet, 0^*(\oplus + \otimes))$, and

(I11) $(\{accb; a, b \in \{\oplus, \otimes\}, c \in \{\circ, \bullet\}\}, \text{Accept})$.

It is easy to see that M' is strictly 4-testable (note that every strictly k-testable language is also a strictly $k+1$ testable language for all $k > 0$ [73]):

- Languages $a0^*$ where $a \in \{\oplus, \otimes\}$ correspond to the triple $(\{a0\}, \{00\}, \{a0, 00\})$.
- Languages Σ_c^* where $c \in \{\circ, \bullet\}$ correspond to the triple $(\Sigma_c, \Sigma_c, \Sigma_c)$.
- Languages $\Sigma_c^* a$ where $c \in \{\circ, \bullet\}$ and $a \in \{\oplus, \otimes\}$ correspond to the triple $(\Sigma_c^2, \Sigma_c^2, \Sigma_c \cdot \{a\})$.
- Language $0^*(\oplus + \otimes)$ corresponds to the triple $(\{0\oplus, 0\otimes, 00\}, \{00\}, \{0\oplus, 0\otimes\})$.
- Language $\{accb; a, b \in \{\oplus, \otimes\}, c \in \{\circ, \bullet\}\}$ corresponds to the triple $(\{\oplus \circ \circ, \oplus \bullet \bullet, \otimes \circ \circ, \otimes \bullet \bullet\}, \emptyset, \{\circ \circ \oplus, \circ \circ \otimes, \bullet \bullet \oplus, \bullet \bullet \otimes\})$.
- All rewritten subwords are shorter than 5 symbols.

Let us prove that $L = \mathrm{L}(M')$. We will start with $L \subseteq \mathrm{L}(M')$:

- The words shorter than 5 symbols are accepted by the accepting meta-instruction and they will not be considered in the following cases.
- The words from L_1 are reduced by (I1)–(I4) to a word from L_3.
- The words from L_2 are reduced by (I5)–(I8) to a word from L_4.
- The words from $L_3 \cup L_4$ are reduced by (I9) and (I10) to a word from $L_1 \cup L_2$.

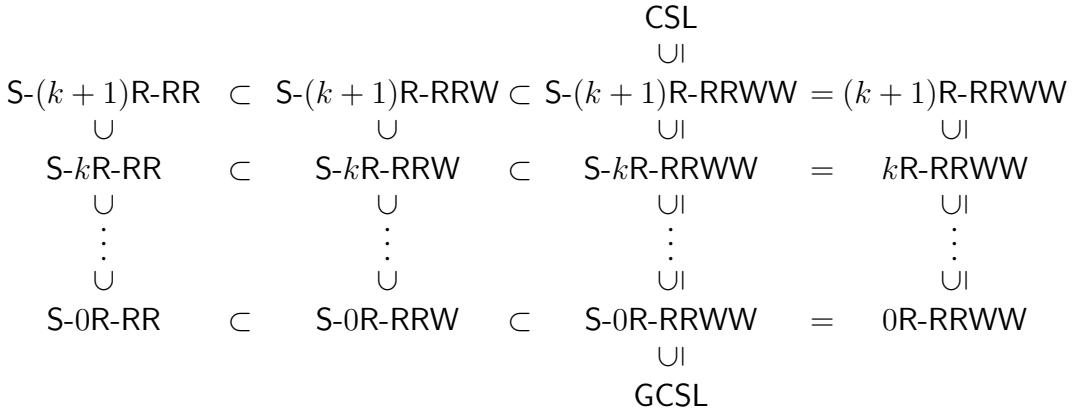


Figure 6.14: Important relations regarding languages accepted by several kinds of restarting automata which hold for all $k \geq 1$.

Thus, each word from L is either accepted by M' directly or reduced into a shorter one. Thus, $L \subseteq \mathrm{L}(M')$.

Now let us prove that $\mathrm{L}(M') \subseteq L$. We will start with proving that no rewriting meta-instruction reduces a word outside L to a word from L.

- Let us consider the meta-instructions (I1)–(I8). Let $w' \in L$ be a word obtained by applying one of those meta-instructions. By reverting the rewrite, we can easily obtain the original word that surely belongs to L as well.
- Let us consider the meta-instructions (I9) and (I10). Let $w' \in L$ be a word obtained by applying one of those meta-instructions. By reverting the rewrite, we can easily obtain the original word that surely belongs to L as well.
- The accepting meta-instruction (I11) accepts exactly the listed words that all belong to L.

Thus, $L = \mathrm{L}(M')$. This concludes our proof.

□

We proved several theorems about languages accepted by several kinds of restarting automata. In Fig. 6.14 we review some of the important relations. For brevity, we write e.g. S-kR-RR instead of $\mathcal{L}$(S-kR-RR) to denote the class of languages accepted by S-kR-RR-automata. The relations hold for $k \geq 1$. The symbols $\cup$ and $\cup\!|$ are just rotated symbols $\subset$ and $\subseteq$, respectively. The inclusion $\mathcal{L}$(S-kR-RRWW) $\subseteq$ CSL follows from the fact that each restarting automaton can be simulated by a linear bounded automaton. The remaining relations come from Theorem 6.2.9, Theorem 6.4.7, Theorem 6.4.8, Theorem 6.4.9, and Theorem 6.4.10.

Below we present a small generalization of the inference algorithm Omega2. While Omega2 can infer S-ZR-RRWW-automata, the new method Omega* can infer S-kR-RRWW-automata for any $k \geq 0$. By 0-RI we denote the algorithm ZR.

Algorithm 6.4.13. Omega* method.
Input: a finite set of FPABR samples S, a reversibility level k to be considered, an input alphabet Σ, and a working alphabet Γ.
Output: a S-kR-RRWW-automaton consistent with S.

1. If S is empty, then stop and return $M = (\Sigma, \Gamma, \emptyset)$.

2. Transform the set S into the set of located reductions (using the first-opportunity rewritings) $R = \{(u_i, v_i \to v'_i, w_i); i \in I\}$.

3. Transform each maximal group $\{(u_i, v \to v', w_i); i \in I'\} \subseteq R$ of compatible reductions into the rewriting meta-instruction

$$(k\text{-RI}(\{u_i; i \in I'\}), v \to v', k\text{-RI}(\{w_i; i \in I'\}))$$

   to obtain the set of rewriting meta-instructions RW.

4. Transform the set B of simple words from samples S into the accepting meta-instruction $accept = (k\text{-RI}(B), \text{Accept})$.

5. Return the restarting automaton $M = (\Sigma, \Gamma, RW \cup \{accept\})$.

We denote the output of Omega* with Omega$^*(k, S, \Sigma, \Gamma)$ or simply with Omega$^*(k, S)$ (if the alphabets are obvious for the particular context).

Again, it is easy to see that the following holds.

Theorem 6.4.14. *Given a finite set of FPABR samples S and a reversibility level k,* Omega* *returns a* S-kR-RRWW*-automaton consistent with S.*

Clearly, increasing the reversibility level in Omega* leads, generally, to inferring smaller languages.

Theorem 6.4.15. *Let S be a finite set of FPABR samples and $k \geq 0$. It holds* $\text{L}(\text{Omega}^*(k+1, S)) \subseteq \text{L}(\text{Omega}^*(k, S))$.

Proof. The only difference in the run of Omega* for different values of k is in the process of inferring the contexts and the base language. Note that they are inferred from the same samples for any considered reversibility level. According to the Theorem 6.4.5, the languages obtained in the case of reversibility level equal to $(k+1)$ are subsets of corresponding languages inferred in the case of reversibility level equal to k (note that k-reversible languages are also $(k+1)$-reversible). Obviously, each word accepted by the automaton Omega$^*(k+1, S)$ is accepted by the automaton Omega$^*(k, S)$ as well.

□

The Omega* method allows the teacher to tune the reversibility level as needed to achieve better results. We hope it will sometimes help to avoid overgeneralization occurring at low reversibility levels. According to the results presented in Chapter 7, this method performs significantly better compared to its predecessor, Omega2, in benchmarks using random targets.

7. Testing

This chapter presents a benchmark used to evaluate the algorithm Omega* proposed in Chapter 6. An evaluation of grammatical inference algorithm is far from obvious task. Target languages as well as inferred languages are in general infinite sets, hence, their comparison can be hard. In Section 7.1 we introduce several kinds of benchmarks used by other researchers. Problems related to the evaluation of restarting automata inference methods are presented in Section 7.2 where also a new benchmark for evaluation of our method is proposed. Results achieved by the Omega* method are presented in Section 7.3.

7.1 Approaches to Testing

There are several approaches to evaluate inference algorithms. Sometimes, we are able to prove some theorem describing the performance of an algorithm on some class of languages. However, proving such theorems can be quite hard. Fortunately, there are other useful ways of analyzing performance of learning algorithms. They often consist in running the algorithm on some sample problem input and examining the obtained output.

We will consider the simple case of inferring a language from a given set of sample labeled words for a while. Here some basic benchmarks may have a very similar scenario (let A be the algorithm being evaluated):

1. Choose a target language L.
2. Select a set S of sample words labeled according to their membership in the target language L.
3. Infer a hypothesis H from samples S using the algorithm A.
4. Select another set T of sample words.
5. Label the words in T according to the hypothesis H, and compare the resulting labeling with L to see the performance of the algorithm.

This general scenario rises two questions. How to select target languages? How to select sample words? Several approaches to answering these questions can be found in literature [68, 18, 45].

Some researchers use a fixed set of target languages as well as fixed sets of labeled samples to learn from. A popular example is the set of Tomita's languages [68]. It contains seven regular languages and for each of them, Tomita supplied a small set of labeled sample words. All those languages are quite simple and thus are considered to form a very basic benchmark. Nevertheless, they are used very often. A good reason to do so is to perform some initial tests in order to show that a given method is able to infer some interesting (though easy) languages.

The set of languages defined by Tomita [68] was further extended by Dupont [18] into a set of 15 regular languages. However, in contrast to the previous paragraph, Dupont no longer considers fixed set of sample words but those are generated at random (see [18] for details). He performed several experiments

using the same target language but different sets of sample words to obtain more significant results on the performance of the inference method. For example, in the case of Tomita's languages and the fixed sets of sample words, one can have an inference algorithm that for some reason fails on the fixed samples given by Tomita while it excels in most other cases or vice versa. Introducing randomness and repeated experiments on different data helps to reduce the risk of this to happen. However, note that the set of considered languages remains fixed. On the other hand, the set of fixed languages can assure that we do not omit some of the languages we are interested in.

Following the approach of introducing randomness into benchmarks, we can also select target languages at random. This approach appears e.g. in the Abbadingo One learning competition [45], where target languages for the learning challenge were generated as large random DFSA's. Using target languages generated quickly at random helps us to perform testing on a large number of problem instances. Again, this reduces the chance of using some singular benchmark problems.

7.2 How to Evaluate Restarting Automata Inference Methods

The approaches presented in Section 7.1 consider as a source of information about target languages only sample words. In contrast, our learning algorithm expects a richer input consisting of sample reductions as well. Without them, our inference algorithm would be nearly useless. But note that we introduced sample reductions in order to simplify the hard problem of language inference. In the case of performance tests, it is therefore reasonable to consider input enriched this way too. Thus, we need a new type of benchmark different from those ones presented in Section 7.1.

We know already that for modelling the ABR one can use restarting automata. A method of learning restarting automata from positive samples only was proposed in [55]. However, tests on neither artificial languages nor real ones were presented. In [31] a method of learning from positive data S-ZR-RRWW-automata was presented. The evaluation of the method was done using some manually selected samples (i.e. FPABR samples) on some sample languages used by other researchers (particularly [21]). This testing corresponds to the simplest case among those presented in Section 7.1 — a fixed set of languages and one fixed set of samples for each of the languages. However, when writing that paper the author was not aware of any other suitable way to compare the method presented there with other inference algorithms. There the goal was rather to show that the proposed method was capable of achieving interesting results.

There were some attempts to learn restarting automata using genetic algorithms [35, 12, 32]. In [32] a very limited set of languages was used for tests, where input samples were selected by hand. In [35, 12] an interesting method defining positive and negative reductions based on a given context-free grammar was used. This method can be used for context-free languages to automatically create positive and negative input samples in form of both sets of words and sets of reductions. Though this method could be considered better in some aspects

compared to selecting input samples manually, it has also several drawbacks we will analyze in Subsection 7.2.1. We also present some theoretical limits of similar grammar-based methods. Those results are based on [29].

We consider a testing scenario very similar to that one presented in Section 7.1 (let A denote the algorithm being evaluated):

1. Choose a target language $L \subseteq \{0,1\}^*$.
2. Select an $\overline{\mathrm{R}}$-presentation S of a restarting automaton over $\Gamma = \{0,1\}^*$ according to L.
3. Infer a hypothesis H from samples S using the algorithm A.
4. Select a set $T \subseteq_{\mathrm{fin}} \Gamma^*$ of sample words.
5. Label the words in T according to the hypothesis H, and compare the resulting labeling with L to evaluate the performance of the algorithm.

In order to choose a target language in the first step, we select one of its representations. Usually, different kinds of automata and grammars are used to represent a language. We therefore consider the following two options:

1. grammars and
2. RRWW-automata.

Our inference algorithm deals with S-kR-RRWW-automata. It would be therefore straightforward to choose that type of automata as possible representations of target languages. Actually, we decided to perform the benchmark only for the case of no auxiliary symbols (i.e. we consider inferring S-kR-RRW-automata only), because they play very marginal role in the inference process — they are just copied into the automaton returned by the inference algorithm. However, for representing target languages, we consider the more powerful model of RRW-automata to see the performance on a richer class of languages. Restarting automata form a natural way for obtaining training and testing data for benchmarks we are searching for.

On the other hand, note that in the case of grammars, it is not so obvious what class of grammars to use. Moreover, the transition from a grammar to training and testing data is also not so clear.

Below we briefly analyze both approaches. Grammars are considered in Subsection 7.2.1 and automata are considered in Subsection 7.2.2.

7.2.1 Grammar-Based Evaluation Method

In this subsection we discuss benchmarks working with grammars as representations of target languages. At first, we critically examine the benchmark presented in [35] that defines positive and negative samples directly from a given context-free grammar. Then we also prove that there is a common drawback of all methods starting with linear (or more powerful) grammars. It is the fact that we are not able to decide whether restarting automata need auxiliary symbols in order to accept the language generated by a given grammar. On the other hand, we show

that it is not the case when only *even linear grammars* are considered. At last, we prove that auxiliary symbols are actually needed in order to accept all even linear languages by restarting automata.

Samples Directly from Grammars

Our method only needs positive samples to be present in the input. However, it is useful to think about both kinds of samples — positive as well as negative ones as describing the positive ones immediately gives a description of the negative ones. In [35] a method defining positive and negative reductions based on a given context-free grammar was presented. This method can be used to automatically create positive and negative input samples in form of both sets of words and sets of reductions. We will discuss it now. You can be wondering why we speak about individual reductions and do not consider whole FPABR samples (as they should form input to our inference algorithm). Since reductions are the base of the ABR, there arises a natural question, which reduction samples we consider as positive and which reduction samples as negative. We will show that the approach to positive and negative reduction samples is quite doubtful in this testing approach.

In order to generate training and testing samples, we need to create sample words from the language and possibly also from its complement. From this point of view, the class of context-free grammars seems to be the most suitable class of grammars among those in Chomsky's hierarchy. The more general class of context-sensitive grammars is too complex to be used for generating sample words (even to decide whether a given grammar yields any word is undecidable [48]). On the other hand, the class of regular grammars is much less powerful and thus we do not consider it interesting for our benchmark.

We can use several ways to generate positive and negative sample words based on some given context-free grammar. We are not going to describe possible approaches here as the goal of this subsection is to show why not to start with grammars at all. We will focus on the process of generating positive samples of reductions. At the beginning we should ask ourselves — what is a positive reduction sample? If we have an algorithm for performing the ABR of the target language, for example in the form of a restarting automaton accepting the language L generated by the considered grammar, we have an easy answer. The positive reductions are all reductions performed by that restarting automaton both on words from L and on words from the complement of L. A more interesting case is when we do not have such a restarting automaton to start with, for example when we have only the grammar of the language to be used as the target language. Below we present the approach of [35].

Let us assume that we have a context-free grammar G such that $L = \mathrm{L}(G)$. In [35] samples of positive reductions of the following two types were generated:

1. Positive reductions on words from L: Let $w = u_\ell x u_\mathrm{r} \in L$ and $w' = u_\ell y u_\mathrm{r}$. We consider the located reduction $(u_\ell, x \to y, u_\mathrm{r})$ to be positive if $|x| > |y|$, there is a derivation tree for w where x is the whole word yielded by a subtree rooted with some non-terminal symbol X, and it is possible to derive y from X as well. This situation is depicted in Fig. 7.1.

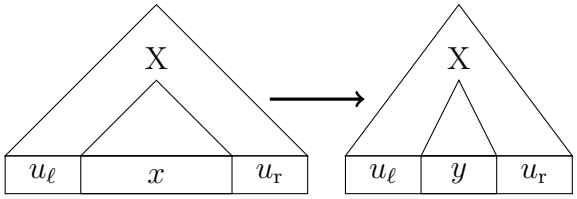


Figure 7.1: A positive reduction of a word $u_\ell x u_\mathrm{r} \in L$.

2. Positive reductions of words out of L: Let $w = u_\ell x u_\mathrm{r} \notin L$ and $w' = u_\ell y u_\mathrm{r}$. We consider the located reduction $(u_\ell, x \to y, u_\mathrm{r})$ to be positive if $w' \notin L$ and there is a positive reduction $(u'_\ell, x \to y, u'_\mathrm{r})$ for some words u'_ℓ, u'_r where $u'_\ell x u'_\mathrm{r} \in L$ and $\mathrm{d}_{\mathrm{edit}}(u_\ell, u'_\ell) + \mathrm{d}_{\mathrm{edit}}(u_\mathrm{r}, u'_\mathrm{r}) = 1$, where $\mathrm{d}_{\mathrm{edit}}$ denotes the edit distance of its arguments [47] (i.e. the minimum number of insert/delete/replace symbol operations needed to transform one word into the other). So the reduction of a word outside L is based on some reduction of some near word from L (if any).

Reductions not falling into the previous cases are called negative.

Though this is a natural way of generating positive reduction samples as it is closely related to the derivation tree of words being reduced, it has serious drawbacks when taking into account that we are trying to infer an RRWW-automaton.

Let us suppose $G = (\{0,1\}, \{S\}, \{S \to 01, S \to 0S1\})$. Thus, it holds $\mathrm{L}(G) = \{0^n 1^n; n \geq 1\}$. Then the above presented approach gives that $(0^n, 0011 \to 01, 1^n), (0^{n+1}, 0011 \to 01, 1^n)$, and $(0^n, 0011 \to 01, 1^{n+1})$ are positive samples of reductions for any given n (the last two samples are positive reductions of words outside L).

On the other hand, when the difference of the number of 0's on the left and the number of 1's on the right is greater than one, the analogous reduction is considered to be negative. Unfortunately, the given samples do not correspond to the ABR that can be modelled by an RRWW-automaton (for us this means the samples do not correspond to the ABR at all). Each rewriting meta-instruction performing one of those positive reductions can perform at most four of them (let $n \geq 0$ be a constant):

- $(0^n, 0011 \to 01, 1^n)$,
- $(0^n, 0011 \to 01, 1^{n+1})$,
- $(0^{n+1}, 0011 \to 01, 1^n)$, and
- $(0^{n+1}, 0011 \to 01, 1^{n+1})$.

The presence of e.g. $(0^{n+2}, 0011 \to 01, 1^{n+2})$ in the list would inevitably mean that the negative reduction $(0^n, 0011 \to 01, 1^{n+2})$ can be performed as well. Thus, in order to perform all reductions considered as positive and at the same time to perform no reduction considered as negative, infinitely many rewriting meta-instructions would be needed. This can be considered to be a big issue of the method. We are not supplying samples corresponding to the ABR that can be modelled by a restarting automaton. Also note that there is a very simple RR-automaton accepting $\mathrm{L}(G)$.

Are Auxiliary Symbols Needed? – Linear Languages

There is also another important thing to note well. Each context-free grammar G can be transformed into an RRWW-automaton (which uses non-terminal symbols of G as auxiliary symbols) accepting $\mathrm{L}(G)$ [37]. It is then easy to give a lot of samples of reductions as shown in Subsection 7.2.2 below. However, the automata obtained in this way are all very simple because the power of auxiliary symbols allows that. We would like to measure the performance on other automata as well, where possible, because, in practice, the ABR using auxiliary symbols for some languages may be unknown in which case the auxiliary symbols will not be used in the input. In practice, we probably do not know the grammar as otherwise we could easily obtain the corresponding RRWW-automaton and there would be no need for learning. Once we know that auxiliary symbols are necessary, it is of course acceptable to consider those simple restarting automata. Nevertheless, when they are not needed, we would like to see the performance on the ABR not using them. However, how can we decide whether any auxiliary symbols are needed? We will analyze this problem below.

In the case of RRW-automata, it could be also much easier to analyze given meta-instructions of automata obtained from the inference algorithm, e.g. to decide whether they are correct and correspond to our idea, despite possibly more complex languages present in meta-instructions. Thus we consider automata limited only to input symbols to be of a particular interest.

Also note that there are no auxiliary symbols in the positive reductions defined according to the above presented approach from [35] no matter what the grammar G is. However, in general, we can not omit auxiliary symbols if we need to accept all context-free languages by restarting automata [37].

In the case of performing tests on a few sample models, one can try to build manually suitable RRW-automata (i.e. not using any auxiliary symbols) for the target grammars and then use those automata to create input samples (thus avoiding using grammars for defining positive and negative sample reductions directly). However, in the case of randomly generated target grammars, it would be nice to decide if there are corresponding RRW-automata automatically (in other words, whether no auxiliary symbols are necessary in order to define restarting automata accepting languages generated by those target grammars). Unfortunately, this poses a problem as described below.

We will prove that the problem whether for a given linear grammar G it is possible to find an RRW-automaton M (having an upper bound on the size of that automaton) such that $\mathrm{L}(G) = \mathrm{L}(M)$ is algorithmically undecidable. Below we will restate the result we presented already in [29]. Consequently, when considering some reasonable upper bound for the size of allowed automata, we are not able to decide this problem at all in practice. In other words, we are not able for a given grammar to design an equivalent RRW-automaton (i.e. accepting the language generated by that grammar) being limited by a given size (when such an automaton exists) or to say that it is not possible otherwise.

We start with a theorem that is auxiliary for the following proofs. This theorem states that the linear grammar universality problem is undecidable [9], i.e. there is no algorithm that takes a linear grammar and decides whether it yields all words over its terminal alphabet.

Theorem 7.2.1 ([9]). *Let* $\Sigma = \{0, 1\}$ *and let the linear grammar universality problem* (F_{LGU}) *be defined as follows:*

Input: a linear grammar $G = (\Sigma, \Gamma, S, P)$.
Output: Does it hold $\mathrm{L}(G) = \Sigma^*$*?*

The F_{LGU} *problem is undecidable.*

We will use this theorem to show some theoretical limits of grammar-based evaluation methods for ABR inference algorithms. As said before, we would like to decide whether for a given linear grammar G it is possible to find an RRW-automaton M such that $\mathrm{L}(G) = \mathrm{L}(M)$. The following theorem states this is undecidable if we allow only RRW-automata of some limited size.

In what follows, we consider only restarting automata having the binary input alphabet $\{0, 1\}$ and having meta-instructions where regular languages are given using regular expressions.

To measure the size of given RRW-automata, we use the so-called AF-measure function. Its definition only slightly limits the class of possible measure functions. At first we define some auxiliary sets.

Definition 7.2.2. *Let* $f : \mathcal{A}(\mathsf{RRW}) \to \mathbb{N}$ *be a function measuring the size of a given automaton, let* $k \in \mathbb{N}$ *be an upper limit on the size of considered automata, and let* $\mathcal{L}$ *be a set of languages.*

- *Let* $L_k^f = \{L;$ *there is an* RRW*-automaton* A *such that* $\mathrm{L}(A) = L$ *and it holds* $f(A) \leq k\}$.
- *We say that a set* $\mathcal{M}$ *of* RRW*-automata has the* $(\mathcal{L}, f, k)$*-property if for all* $M \in \mathcal{M}$ *it holds* $f(M) \leq k$ *and* $\{\mathrm{L}(M); M \in \mathcal{M}\} = \mathcal{L}$.

Below we define the requirements for a function to be an AF-measure function.

Definition 7.2.3. *A function* $f : \mathcal{A}(\mathsf{RRW}) \to \mathbb{N}$ *is called an AF-measure if the following conditions are met:*

(AF1) *there is an algorithm giving the finite set of* RRW*-automata that has the* (L_k^f, f, k)*-property for all* $k \in \mathbb{N}$*, and*

(AF2) *there is an* RRW*-automaton* M *such that* $f(M) = 0$ *and* $\mathrm{L}(M) = \{0, 1\}^*$.

An example of an AF-measure function is $f_{\mathrm{AF}}(M) = \mathsf{size}(M) - 1$, where $\mathsf{size}(M)$ measures the size of automaton M considering the size of FSA's that are used to represent its meta-instructions (see Definition 4.1.3). It is easy to see that f_{AF} is an AF-measure function:

- The set of all RRW-automata (with the input alphabet containing only 0 and 1) having f_{AF} value at most $k \in \mathbb{N}$ can be obtained easily. Note that the number of meta-instructions is limited (we can safely suppose that there is always at most one accepting meta-instruction) as well as the number of languages that can occur in them, thus the resulting set of automata is finite as well. Obviously, all those automata have f_{AF} value at most k and the set of languages accepted by them is exactly $L_k^{f_{\mathrm{AF}}}$, thus this set has the $(L_k^{f_{\mathrm{AF}}}, f_{\mathrm{AF}}, k)$-property. Therefore, the requirement (AF1) is satisfied.

- The RRW-automaton $M = (\{0,1\}, \{0,1\}, \{(\{0,1\}^*, \text{Accept})\})$ accepts the language $\{0,1\}^*$. It has only one meta-instruction that contains a language that can be accepted by a DFSA having exactly one state. Therefore, $\mathsf{size}(M) = 1$ and $f_{\text{AF}}(M) = 0$. Thus, the requirement (AF2) is satisfied.

As said above, in the case of $f_{\text{AF}}(M) = \mathsf{size}(M) - 1$, the set of all automata up to a given size is finite. Note, however, that we allow even measures such that the set of all automata up to a given size is infinite. For example, there can be infinitely many small automata accepting one particular language. However, to satisfy the requirement (AF1), it suffices that one of those automata is present in the set having the (L_k^f, f, k)-property (for all $k \geq 0$ and the given measure f).

Below we prove that it is not possible to decide whether there is a small RRW-automaton accepting language generated by a given linear grammar.

Theorem 7.2.4. *Let f be an AF-measure function. Let $k \in \mathbb{N}$ be an upper limit on the size of considered automata. Let the problem $F_{\text{LIN}-\text{AUX}-f,k}$ be defined as follows:*

Input: a linear grammar G
Output: Does it hold $\text{L}(G) \in L_k^f$?

The problem $F_{\text{LIN}-\text{AUX}-f,k}$ is not decidable.

Proof. We will reduce F_{LGU} to $F_{\text{LIN}-\text{AUX}-f,k}$. Let $G = (\{0,1\}, \Gamma, S, P)$ be a linear grammar. The goal is to decide whether this grammar yields all words over $\{0,1\}$. We distinguish two cases according to the answer to the problem $F_{\text{LIN}-\text{AUX}-f,k}$ on the input G:

- if the answer is *no*, surely $\text{L}(G) \neq \{0,1\}^*$ (because of the (AF2) condition in the definition of the AF-measure function).

- if the answer is *yes*, we know there is an RRW-automaton M such that $\text{L}(M) = \text{L}(G)$. It is easy to get a set of words reduced by a particular rewriting meta-instruction $(L_\ell, x \to y, L_\text{r})$ — it is the set $L_\ell \cdot \{x\} \cdot L_\text{r}$. For a given RRW-automaton $N = (\Sigma_N, \Sigma_N, I_N)$ let

$$\text{K}(N) = \{w \in \Sigma_N^*; N \text{ reduces } w \text{ or accepts it directly}\}$$

  (note it is a regular language). Now we generate all RRW-automata from the set having the (L_k^f, f, k)-property and analyze each particular automaton N using the steps below:

  1. If $\text{K}(N) = \{0,1\}^*$, then remember that fact (as if N accepts $\text{L}(G)$, then the answer for the F_{LGU} problem on G is *yes*), end processing of this automaton, and continue with the next automaton.
  2. Let $w \in \{0,1\}^* \setminus \text{K}(N)$.
  3. If $w \in \text{L}(G)$, then $\text{L}(N) \neq \text{L}(G)$. End processing of N and continue with the next automaton.
  4. Otherwise, $w \notin \text{L}(G)$. Therefore, $\text{L}(G) \neq \{0,1\}^*$ and we stop — the answer for the F_{LGU} problem on G is *no*.

After processing all automata, we either know $\mathrm{L}(G) \neq \{0,1\}^*$ or we know the only candidates for M are the automata accepting all words over $\{0,1\}$. Thus, in the latter case, the answer for the F_{LGU} problem on G is *yes*.

□

We see that even in the case of linear grammars, there are undecidable problems that make it impossible to use the proposed evaluation method.

In Proposition 2.2.2 we introduced the error preserving property of restarting automata. It is useful to consider a similar property regarding individual meta-instructions.

Definition 7.2.5. *Let $M = (\Sigma, \Sigma, I)$ be an* RRW*-automaton and let $L \subseteq \Sigma^*$ be a language. We say that a rewriting meta-instruction $(L_\ell, x \to y, L_\mathrm{r}) \in I$ has the* error preserving property with respect to the language L *if for all $u_\ell \in L_\ell, u_\mathrm{r} \in L_\mathrm{r}$, it holds that $u_\ell x u_\mathrm{r} \notin L$ implies $u_\ell y u_\mathrm{r} \notin L$.*

We can consider another idea. Instead of building a full RRW-automaton from the target grammar, we could try to guess a rewriting meta-instruction at random and then decide if it could be a part of an RRW-automaton accepting the language generated by the considered grammar. Maybe that would not lead to all rewriting meta-instructions needed to parse the target language but let us ignore it for now.

Here one should ask whether we are generally able to decide whether a particular rewriting meta-instruction is error preserving with respect to language of some linear grammar. We prove below that this is impossible as well [29]. Thus, even our limited approach is not suitable for benchmarks.

Theorem 7.2.6. *Let the problem $F_{\mathrm{LIN-EPR}}$ be defined as follows.*

Input: (G, I) where G is a linear grammar and I is a rewriting meta-instruction.
Output: Is I error preserving with respect to $\mathrm{L}(G)$*?*

The problem $F_{\mathrm{LIN-EPR}}$ is undecidable.

Proof. We will reduce the problem F_{LGU} to the problem $F_{\mathrm{LIN-EPR}}$. Let $\Sigma = \{0,1\}$ and let $G = (\Sigma, \Gamma, S, P)$ be a linear grammar that forms the input for the problem F_{LGU}. Let $I_0 = (\Sigma^*, 0 \to \lambda, \Sigma^*)$ and $I_1 = (\Sigma^*, 1 \to \lambda, \Sigma^*)$. We distinguish two cases according to the answers for the problem $F_{\mathrm{LIN-EPR}}$ on the inputs (G, I_0) and (G, I_1):

a) If for both inputs the answer for the problem $F_{\mathrm{LIN-EPR}}$ is *yes*, then we can reduce any non-empty word w by removing any of its symbols until we get the empty word using I_0 and I_1. If $\lambda \in \mathrm{L}(G)$, then we know (because of the error preserving property of I_0 and I_1 with respect to $\mathrm{L}(G)$) that $w \in \mathrm{L}(G)$ as well and thus $\mathrm{L}(G) = \{0,1\}^*$ and so the answer for the F_{LGU} problem on G is *yes*. If $\lambda \notin \mathrm{L}(G)$, then surely $\mathrm{L}(G) \neq \{0,1\}^*$ and the answer is therefore *no*.

b) On the other hand, if there is $i \in \{0,1\}$ such that the answer for the $F_{\mathrm{LIN-EPR}}$ problem on (G, I_i) is *no*, then we know that the particular meta-instruction is not error preserving with respect to $\mathrm{L}(G)$. So there is a word w such that $w \notin \mathrm{L}(G)$ (and it can be reduced to a word from $\mathrm{L}(G)$ — but this part of the definition is not important here). Thus, $\mathrm{L}(G) \neq \{0,1\}^*$ and therefore the answer for the F_{LGU} problem on G is *no*.

□

Are Auxiliary Symbols Needed? - Even Linear Languages

We can not decide whether for a given linear grammar there exists a small RRW-automaton accepting language generated by that grammar. Hence, let us consider even more restricted version of context-free grammars, the so-called even linear grammars.

Definition 7.2.7 ([2, 67]). *A grammar* (Σ, Γ, S, P) *is an* even linear grammar *if each rule from* P *has one of the following forms (where* $X, Y \in \Gamma, a, b \in \Sigma$*):*

- $X \to aYb$,
- $X \to a$, *or*
- $X \to \lambda$.

A language L *is an* even linear language *if there is an even linear grammar* G *such that* $\mathrm{L}(G) = L$.

We denote the class of even linear languages by EL.

Usually, the even linear grammars are defined so that they allow slightly more general rewriting rules. Then the definition introduced above describes the so-called normal form of even linear grammars [67]. In this thesis we work only with grammars in that normal form and we will call them just even linear grammars as defined above.

There is a simple transformation (the so-called *folding*) that enables us to relate even linear languages to regular languages.

Definition 7.2.8 ([66]). *Let* Σ *be an alphabet. We define* $\Sigma_{\mathrm{f}} = \{\frac{a}{b}; a, b \in \Sigma\}$. *For* $w \in \Sigma^*$, *we define* $w^{\mathrm{f}} = w$ *if* $|w| \leq 1$ *and* $(awb)^{\mathrm{f}} = \frac{a}{b}w^{\mathrm{f}}$ *otherwise. Let* $L \subseteq \Sigma^*$ *be a language. We define* $L^{\mathrm{f}} = \{w^{\mathrm{f}}; w \in L\}$ *and we call* L^{f} *the* folding of L.

The following theorem introduces a relation between even linear languages and regular languages.

Theorem 7.2.9 ([46]). *A language* L *is even linear if and only if* L^{f} *is regular.*

To describe the language L^{f}, we can use the folding of a grammar as defined below.

Definition 7.2.10 ([2]). *The* folding *of an even linear grammar* $G = (\Sigma, \Gamma, S, P)$ *is the regular grammar* $G_{\mathrm{f}} = (\Sigma \cup \Sigma_{\mathrm{f}}, \Gamma, S, P_{\mathrm{f}})$, *where* P_{f} *is* $\{A \to \frac{a}{b}B; A \to aBb \in P\} \cup \{A \to a; A \to a \in P\} \cup \{A \to \lambda; A \to \lambda \in P\}$.

Having this definition, we can relate the language of the grammar folding to the language folding.

Theorem 7.2.11 ([2]). $\mathrm{L}(G_{\mathrm{f}}) = \mathrm{L}(G)^{\mathrm{f}}$ *for all even linear grammars* G.

It is an important result that the class of even linear languages properly contains the class of regular languages.

Theorem 7.2.12 ([2, 50])**.** *The class of even linear languages properly contains the class of regular languages.*

Proof. Let L be a regular language accepted by some DFSA $A = (Q, \Sigma, \delta, \{q_0\}, F)$. We will create an even linear grammar G such that $\mathrm{L}(G) = \mathrm{L}(A)$. Let $G = (\Sigma, \Gamma, S, P)$, where

- $\Gamma = \{S_{r,R}; r \in Q, R \subseteq Q\}$ is the set of non-terminal symbols,
- $S = S_{q_0,F}$ is the initial non-terminal symbol, and
- P is the set of rules containing (let $r \in Q, R \subseteq Q, a \in \Sigma$)
  - (R1) $S_{r,R} \to aS_{r',R'}b$ for $r' = \delta(r, a)$ and $R' = \bigcup_{s \in R} \delta^{\mathrm{R}}(s, b)$ (for $R = \emptyset$ let $R' = \emptyset$),
  - (R2) $S_{r,R} \to \lambda$ for $r \in R$, and
  - (R3) $S_{r,R} \to a$ for $\delta(r, a) \in R$.

Let us analyze a derivation of a word according to G. Let us suppose $S \Rightarrow^* uS_{r,R}v$ where $u, v \in \Sigma^*$ and $S_{r,R} \in \Gamma$. Obviously the following holds:

- $r = \delta(q_0, u)$, and
- $R = \{q \in Q; \delta^*(q, v) \in F\}$.

Now it is easy to see that $\mathrm{L}(G) \subseteq L$ as the rules (R2) and (R3) use the information stored in the non-terminals to finish the derivation only when a word from L will be obtained by applying the particular rule.

On the other hand, let $uv \in \mathrm{L}(A)$ where $u, v \in \Sigma^*$ such that $|u| = |v|$. Obviously, $S \Rightarrow^* uS_{r,R}v$ for some $S_{r,R} \in \Gamma$ and it holds $r \in R$. Therefore, the rule (R2) finishes the derivation and uv is obtained. Similarly, let $uav \in \mathrm{L}(A)$ where $a \in \Sigma, u, v \in \Sigma^*$ such that $|u| = |v|$. Again, $S \Rightarrow^* uS_{r,R}v$ for some $S_{r,R} \in \Gamma$. This time it holds $\delta(r, a) \in R$ and thus the rule (R3) finishes the derivation and uav is obtained. Thus, $\mathrm{L}(A) \subseteq \mathrm{L}(G)$.

Overall, it holds $\mathrm{L}(G) = L$.

To see that the inclusion is proper, consider the obviously even linear language $\{0^n 1^n; n > 0\}$.

□

We will use the following properties of even linear languages.

Theorem 7.2.13. *Let Σ be an alphabet and $L, L' \subseteq \Sigma^*$ be two even linear languages. Then $L \cap L'$, $L \cup L'$, and L^{c} are even linear languages.*

Proof. We will present the proof for the case of L^{c}. A similar approach can be used to prove the remaining cases.

Let $G = (\Sigma, \Gamma, S, P)$ be an even linear grammar generating L. We will show how to compute an even linear grammar generating $(\mathrm{L}(G))^{\mathrm{c}}$. At first, we transform G into G_{f} that generates the regular language $\mathrm{L}(G_{\mathrm{f}})$. Therefore, $L_{\mathrm{f,c}} = (\mathrm{L}(G_{\mathrm{f}}))^{\mathrm{c}}$ is a regular language as well. However, $L_{\mathrm{f,c}}$ may contain words that are not a folded version of any word over Σ. Therefore, we define $L'_{\mathrm{f,c}} = L_{\mathrm{f,c}} \cap ((\Sigma^*)^{\mathrm{f}})$. This is a regular language, too. Let $G'_{\mathrm{f,c}}$ be a regular

grammar such that $\mathrm{L}(G'_{\mathrm{f,c}}) = L'_{\mathrm{f,c}}$. This grammar can be easily converted into a regular grammar $G''_{\mathrm{f,c}}$ such that $\mathrm{L}(G''_{\mathrm{f,c}}) = \mathrm{L}(G'_{\mathrm{f,c}})$ and the grammar contains only rules having form $X \to aY, X \to b$, and $X \to \lambda$, where X, Y are non-terminal symbols and $a \in \Sigma_{\mathrm{f}}, b \in \Sigma$ are terminal symbols. It is then easy to convert $G''_{\mathrm{f,c}}$ into an even linear grammar G_{c} such that $G''_{\mathrm{f,c}}$ is a folding of G_{c}. Then, according to Theorem 7.2.11, it holds $\mathrm{L}(G''_{\mathrm{f,c}}) = \mathrm{L}(G_{\mathrm{c}})^{\mathrm{f}}$. Now it is easy to see that $\mathrm{L}(G_{\mathrm{c}}) = (\mathrm{L}(G))^{\mathrm{c}}$ as follows (let $w \in \Sigma^*$):

- $w \in \mathrm{L}(G) \Leftrightarrow w^{\mathrm{f}} \in \mathrm{L}(G)^{\mathrm{f}}$ (obviously),
- $w^{\mathrm{f}} \in \mathrm{L}(G)^{\mathrm{f}} \Leftrightarrow w^{\mathrm{f}} \in \mathrm{L}(G_{\mathrm{f}})$ (according to Theorem 7.2.11),
- $w^{\mathrm{f}} \in \mathrm{L}(G_{\mathrm{f}}) \Leftrightarrow w^{\mathrm{f}} \notin L'_{\mathrm{f,c}}$ (the word w^{f} surely belongs to $(\Sigma^*)^{\mathrm{f}}$),
- $w^{\mathrm{f}} \notin L'_{\mathrm{f,c}} \Leftrightarrow w^{\mathrm{f}} \notin \mathrm{L}(G''_{\mathrm{f,c}})$ (obviously),
- $w^{\mathrm{f}} \notin \mathrm{L}(G''_{\mathrm{f,c}}) \Leftrightarrow w^{\mathrm{f}} \notin \mathrm{L}(G_{\mathrm{c}})^{\mathrm{f}}$ (according to Theorem 7.2.11),
- $w^{\mathrm{f}} \notin \mathrm{L}(G_{\mathrm{c}})^{\mathrm{f}} \Leftrightarrow w \notin \mathrm{L}(G_{\mathrm{c}})$, and, in total, we have
- $w \in \mathrm{L}(G) \Leftrightarrow w \notin \mathrm{L}(G_{\mathrm{c}})$.

□

The following auxiliary theorem [29] will be used in several proofs below.

Theorem 7.2.14. *Let $G = (\Sigma, \Gamma, S, P)$ be an even linear grammar. Let $x, y \in \Sigma^*$ such that $|x| \geq |y|$. Then there is an even linear grammar G' such that $\mathrm{L}(G') = \{uxv; uyv \in \mathrm{L}(G)\}$.*

Proof. At first, we will suppose that $|y| > 0$. The case of $|y| = 0$ will be proved at the end of this proof.

It suffices to consider $|x| = |y|$ and $|x| = |y| + 1$. The cases when $|x| > |y| + 1$ can be solved by applying this theorem several times — at first, we use it to replace y with the word $\diamond^{|y|}$ (where $\diamond$ is a new terminal symbol not present in Σ), then to replace $\diamond^{|y|}$ with $\diamond^{|y|+1}$, then to replace $\diamond^{|y|+1}$ with $\diamond^{|y|+2}$, ..., until we reach $\diamond^{|x|}$, and then we replace $\diamond^{|x|}$ with x.

W.l.o.g. we suppose that $x \in \Sigma^*$ to simplify this proof.

Let us start with $|x| = |y|$. We will modify G so that at some random point while deriving a word, it starts to generate x instead of the output of the original grammar G (i.e. not necessarily instead of y). To generate x only on the places where G generates y, we remember the output of the original grammar in the non-terminal symbol and finish the derivation successfully only when y is remembered. The transformation is done as follows (in what follows, it holds $a, b, c, d \in \Sigma, u, v, w \in \Sigma^*, X, Y \in \Gamma$):

For each $X \in \Gamma$, we add new non-terminal symbols (let them all form the set Γ'; note the domains for u, v, w need to contain only subwords of x and y and thus the number of non-terminal symbols is finite; the exact domains are omitted for brevity):

- X' denoting x has been already generated instead of y in the current derivation.

- $X_v^{\blacktriangleleft,u}$ denoting a non-empty prefix of the word x has been generated to the left of this non-terminal symbol, it remains to generate the word u in order to generate the whole x, and the original grammar G would generate the word v instead of the prefix there.

- $X_v^{\blacktriangleright,u}$ denoting a non-empty suffix of the word x has been generated to the right of this non-terminal symbol, it remains to generate the word u in order to generate the whole x, and the original grammar G would generate the word v instead of the suffix there.

- $X_{(v,w)}^{\blacktriangleleft\blacktriangleright,u}$ denoting both a non-empty prefix and a non-empty suffix of x have been generated to the left and to the right of this non-terminal symbol, it remains to generate u in order to generate the whole x, and the original grammar G would generate the word v instead of the prefix and the word w instead of the suffix there.

We divide a derivation of a word into three phases. In the first one, we derive according to the original rules from P. Then we enter the second phase at some point and start generating x. After x is generated, we finish the derivation in the third phase using nearly the same rules as those present in P.

Let $P_{\text{initial}} = \{(l \to r) \in P; r \text{ contains a non-terminal symbol}\}$. To start generating the word x at any point, we add for each rule $X \to aYb \in P$ the following three rules (let them form a set P_{start}):

- to start generating the word x to the left of the current non-terminal symbol: $X \to cY_a^{\blacktriangleleft,u}b$, where $x = cu$,

- to start generating the word x to the right of the current non-terminal symbol: $X \to aY_b^{\blacktriangleright,u}d$, where $x = ud$,

- to start generating the word x to both the left and the right of the current non-terminal symbol: $X \to cY_{(a,b)}^{\blacktriangleleft\blacktriangleright,u}d$, where $x = cud$.

For each rule $X \to y \in P$, the rule $X \to x$ is added to finish the derivation in this special case where y is generated in one step (let those new rules form the set P_{one}).

After starting the process of generating x, we need to generate the remaining symbols of x. To do so, we add the following rules (let them form a set P_{follow}) for each rule $X \to aYb \in P$:

- continuing to the left of the current non-terminal symbol: $X_v^{\blacktriangleleft,cu} \to cY_{va}^{\blacktriangleleft,u}b$; analogically for $\blacktriangleright$,

- continuing to both the left and the right of the current non-terminal symbol after generating to the left only or to the right only: $X_v^{\blacktriangleleft,cud} \to cY_{(va,b)}^{\blacktriangleleft\blacktriangleright,u}d$; analogically for the case with $\blacktriangleright$,

- continuing to both the left and right of the current non-terminal symbol: $X_{(v,w)}^{\blacktriangleleft\blacktriangleright,cud} \to cY_{(va,bw)}^{\blacktriangleleft\blacktriangleright,u}d$.

Next, we add finishing rules that end generating the word x, let them form a set P_{finish}. We add the following rules for each rule $X \to aYb \in P$:

- $X_y^{\blacktriangleleft,\lambda} \to aY'b$; analogically for the case with $\blacktriangleright$,

and the following rules for each rule $X \to a \in P$:

- $X_y^{\blacktriangleleft,\lambda} \to a$; analogically for the case with $\blacktriangleright$,
- $X_{y'}^{\blacktriangleleft,c} \to c$ where $y = y'a, y' \in \Sigma^*$, analogically for the case with $\blacktriangleright$,
- $X_{(y',y'')}^{\blacktriangleleft\blacktriangleright,c} \to c$ where $y = y'ay'', y', y'' \in \Sigma^*$,

and the following rules for each rule $X \to \lambda \in P$:

- $X_y^{\blacktriangleleft,\lambda} \to \lambda$; analogically for the case with $\blacktriangleright$ and $\blacktriangleleft\blacktriangleright$.

Finally, let $P' = \{X' \to aY'b; X \to aYb \in P\} \cup \{X' \to a; X \to a \in P\} \cup \{X' \to \lambda; X \to \lambda \in P\}$.

The resulting grammar is $G' = (\Sigma, \Gamma \cup \Gamma', S, P_{\text{initial}} \cup P_{\text{start}} \cup P_{\text{one}} \cup P_{\text{follow}} \cup P_{\text{finish}} \cup P')$. It is easy to see that $\mathrm{L}(G') = \{uxv; uyv \in \mathrm{L}(G)\}$:

- The work of G' is analogous to that of G. The main difference is that somewhere in derivations we generate x instead of y — the words stored in the non-terminal symbols ensure that the replacement does not occur elsewhere. Therefore, $\mathrm{L}(G') \subseteq \{uxv; uyv \in \mathrm{L}(G)\}$.
- Let $uyv \in \mathrm{L}(G)$. We will show how to derive uxv using G'. The initial part of the derivation, where only symbols from u and v are generated, is the same as in G (we use rules from P_{initial}). This way, we obtain $u'Xv'$ for some $u'v' \in \Sigma^*, X \in \Gamma$. It remains to generate x and the remaining part of at most one of u and v. If $uyv = u'yv'$ and $|y| = 1$, then we use the corresponding rule from P_{one}. Otherwise, we start generating x by using a rule from P_{start} according to the actual position of x in uxv. Then we generate the following symbols of x and also possibly of u or v using the rules from P_{follow}. For each rule of G having the form $X \to aYb \in P$ that generates a part of y in the derivation of uyv, there will be one derivation step using those rules. Then we use a rule from P_{finish} corresponding to the next step of generating uyv and possibly also some rules from P'. This yields uxv. Therefore, $\{uxv; uyv \in \mathrm{L}(G)\} \subseteq \mathrm{L}(G')$.

Now we will analyze the case when $|x| = |y| + 1$. At first, we will use the previous case to replace y with the word $\#_1 \cdots \#_{|y|}$ where $\#_i$ are new terminal symbols. Thus, let $G' = (\Sigma \cup \{\#_1, \ldots, \#_{|y|}\}, \Gamma', S, P')$ be an even linear grammar such that $\mathrm{L}(G') = \{u\#_1 \cdots \#_{|y|}v; uyv \in \mathrm{L}(G)\}$. We search for all rules having $\#_{|y|}$ on the right-hand side.

Below we create an even linear grammar G'' such that

$$\mathrm{L}(G'') = \{u\#_1 \cdots \#_{|y|}\#_{|y|+1}v; u\#_1 \cdots \#_{|y|}v \in \mathrm{L}(G')\}$$

where $\#_{|y|+1}$ is a new terminal symbol.

Let $P''_{\text{init}} = \{l \to r \in P'; r \text{ contains a non-terminal symbol}\}$.

Next, we add the following rules (let them form a set P''_{tail} and let the new non-terminal symbols form a set Γ''):

- The rule $X \to \#_{|y|}$ is replaced with two rules $X \to \#_{|y|}X'\#_{|y|+1}$ and $X' \to \lambda$, where X' is a new non-terminal symbol.
- In the case of $X \to \#_{|y|}Yb$, we postpone generating the terminal symbols in the following way. For each $X \in \Gamma$, we add a non-terminal symbol $X_{\blacktriangleleft,a}$ denoting the symbol a is postponed. The rule is replaced with $X \to \#_{|y|}Y_{\blacktriangleleft,\#_{|y|+1}}b$. In addition, there are the following new rules to handle the postponing:
  - For each rule $X \to aYb$, there is a new rule $X_{\blacktriangleleft,c} \to cY_{\blacktriangleleft,a}b$.
  - For each rule $X \to a$, there are new rules $X_{\blacktriangleleft,c} \to cY'a$ and $Y' \to \lambda$ (Y' is a new non-terminal symbol).
  - For each $X \to \lambda$, there is a new rule $X_{\blacktriangleleft,c} \to c$.
- The case of $X \to aY\#_{|y|}$ is handled analogically.

Then let the grammar $G'' = (\Gamma' \cup \Gamma'', \Sigma \cup \{\#_1, \ldots, \#_{|y|+1}\}, S, P_{\text{init}} \cup P_{\text{tail}})$. Obviously, the grammar G'' generates exactly all words that can be obtained from words generated by G' by inserting $\#_{|y|+1}$ right after $\#_{|y|}$.

Finally, we replace all $\#_i$ in the definition of G'' with corresponding x_i to obtain the grammar G''' such that $\mathrm{L}(G''') = \{uxv; uyv \in \mathrm{L}(G)\}$. This finishes the case $|x| = |y| + 1$.

It remains to handle the special case of $y = \lambda$. If $x = \lambda$ as well, let $G' = G$. If $|x| > 0$, we w.l.o.g. suppose that $|x| = 1$ and we perform the following steps:

- For each $a \in \Sigma$, we create even linear grammars $G_{a,\ell}$ and $G_{a,\mathrm{r}}$ such that $\mathrm{L}(G_{a,\ell}) = \{uxav; uav \in \mathrm{L}(G)\}$ and $\mathrm{L}(G_{a,\mathrm{r}}) = \{uaxv; uav \in \mathrm{L}(G)\}$ (using this theorem for the case $|y| > 0$). These grammars can be used to obtain words where x is inserted before or after an occurrence of a in a word from $\mathrm{L}(G)$.
- If $\lambda \in \mathrm{L}(G)$, then we create the even linear grammar $G_\lambda = (\{x\}, \{S\}, S, \{S \to x\})$. This grammar can be used to obtain the word where x is inserted anywhere into the empty word.
- The grammars can be easily merged together to obtain an even linear grammar G' satisfying the requirement of the theorem.

□

Below we will show some positive results related to even linear languages. Let us start with partitioning reductions into four classes C1,...,C4. Let G be an even linear grammar, $L = \mathrm{L}(G)$. There are four cases to be considered for a particular located reduction $(u, x \to y, v)$ based on membership of uxv and uyv in L:

(C1) $uxv \in L$ and $uyv \in L$,

(C2) $uxv \in L$ and $uyv \notin L$,

(C3) $uxv \notin L$ and $uyv \in L$ (those are the only reductions that do not preserve errors with respect to L, i.e. they reduce a word outside L to a word from L), and

(C4) $uxv \notin L$ and $uyv \notin L$.

We will show how to describe the classes using even linear grammars. Let us study the general case, so let $L_0, L_1 \in \{L, L^c\}$. The goal is to find a description of the domain for (u, v) such that $uxv \in L_0$ and $uyv \in L_1$.

Thanks to Theorem 7.2.13, we can suppose we have even linear grammars for both L_0 and L_1, let us denote them as G_0 and G_1.

We are going to describe corresponding u's and v's using some operations on grammars and languages. The idea is to somehow *intersect* the sets $\{uxv; uxv \in L_0\}$ and $\{uyv; uyv \in L_1\}$ and then to extract the common contexts for x and y. Also note that there can be several factorizations of a particular word to uxv or uyv for given words x, y.

We will modify the grammars G_0 and G_1 (as shown in the proof of Theorem 7.2.14) to obtain even linear grammars G'_0 and G'_1 such that $\mathrm{L}(G'_0) = \{u \bullet^{|x|} v; uxv \in L_0\}$ and $\mathrm{L}(G'_1) = \{u \bullet^{|x|} v; uyv \in L_1\}$. Note that we have used the same number of symbols $\bullet$ (a new terminal symbol) in both cases. So finally we have $(uxv \in L_0 \text{ and } uyv \in L_1) \Leftrightarrow (u \bullet^{|x|} v \in \mathrm{L}(G'_0) \text{ and } u \bullet^{|x|} v \in \mathrm{L}(G'_1)) \Leftrightarrow (u \bullet^{|x|} v) \in \mathrm{L}(G'_0) \cap \mathrm{L}(G'_1) \Leftrightarrow u \bullet^{|x|} v \in \mathrm{L}(G'')$ for some even linear grammar G'' that can be obtained according to the proof of Theorem 7.2.13.

So for any condition from C1 to C4, we are able to describe all contexts u and v such that the particular condition holds using an even linear grammar. For a given even linear grammar G, rewrite $x \to y$ and an even linear grammar G'' resulting from the just presented approach for the case Ci, let $L^{Ci}_{G,x\to y} = \mathrm{L}(G'')$.

Note that this does not mean we are able to give a standard rewriting meta-instruction performing the reductions described with C1, C2, C3, or C4. Here the contexts are described as a pair using an even linear grammar, but for specifying a rewriting meta-instruction we need two independent regular contexts. The mapping from G'' to rewriting meta-instructions need not be obvious. For example, think of the language $L = \{0^n1^n; n \geq 0\}$ (generated by a grammar G) and the reductions of the form $(u, 01 \to \lambda, v)$. Let us consider the condition C1 as it seems to be sufficient to reduce only the words from the language and to reduce them to words from the language as well. What are the contexts u and v such that the condition C1 holds? We will show that $L^{C1}_{G,01\to\lambda} = \{0^n \bullet\bullet 1^n; n \geq 0\}$:

Let $u \bullet \bullet v \in L^{C1}_{G,01\to\lambda}$. Then surely $uxv = 0^n011^n \in L$ and $uyv = 0^n1^n \in L$ as well. On the other hand, let $uxv = u \cdot 01 \cdot v \in L$. Thus, $u = 0^n$ and $v = 1^n$ for some $n \geq 0$. Thus $u \cdot \bullet \bullet \cdot v = 0^n \bullet\bullet 1^n \in L^{C1}_{G,01\to\lambda}$.

So we have a description of all contexts corresponding to the condition C1. However, it is impossible to give a rewriting meta-instruction performing exactly the described reductions as it would be necessary to ensure that the number of 0's and the number of 1's match, which is not possible to achieve using independent contexts.

On the other hand, we can use the description of contexts to decide whether a particular rewriting meta-instruction is error preserving with respect to the language generated by a given even linear grammar as stated in the following theorem [29].

Theorem 7.2.15. *Let the problem $F_{\mathrm{EL-EPR}}$ be defined as follows.*

Input: an even linear grammar G, a rewriting meta-instruction I
Output: Is I error preserving with respect to $\mathrm{L}(G)$*?*

The problem $F_{\mathrm{EL-EPR}}$ is decidable.

Proof. Let G be an even linear grammar and $I = (L_\ell, x \to y, L_\mathrm{r})$ be a rewriting meta-instruction. It suffices to decide whether $L^{C3}_{G,x\to y} \cap (L_\ell \cdot \{\bullet^{|x|}\} \cdot L_\mathrm{r}) = \emptyset$. This is decidable, as the intersection of a context-free language and a regular language is context-free and emptiness is decidable for all context-free languages.

□

In the case of even linear grammars, even the question whether the language generated by a given grammar G belongs to L^f_k is decidable [29].

Theorem 7.2.16. *Let f be an AF-measure function. Let $k \in \mathbb{N}$. Let the problem $F_{\mathrm{EL-AUX-}f,k}$ be defined as follows.*

Input: an even linear grammar G
Output: Does it hold $\mathrm{L}(G) \in L^f_k$?

The problem $F_{\mathrm{EL-AUX-}f,k}$ is decidable.

Proof. Let G be an even linear grammar. Let $\mathcal{M}$ be the finite set of automata with (L^f_k, f, k) property (remember there is an algorithm enumerating this finite set). Let $\mathcal{M}_0 = \{M; M \in \mathcal{M}$ has only error preserving rewriting meta-instructions with respect to $\mathrm{L}(G)$ and no word outside $\mathrm{L}(G)$ is accepted by M directly$\}$ (note we are able to create this set using Theorem 7.2.15). Thus, for each $M \in \mathcal{M}_0$, we have $\mathrm{L}(M) \subseteq \mathrm{L}(G)$.

With each particular automaton $M = (\Sigma, \Gamma, I)$ from $\mathcal{M}_0$ we perform the following action. We take each rewriting meta-instruction $i = (L_\ell, x \to y, L_\mathrm{r}) \in I$, transform it into the language $L_i = L_\ell \cdot \{\bullet^{|x|}\} \cdot L_\mathrm{r}$, and let us denote $C_i = L_i \cap L^{\mathrm{C1}}_{G,x\to y}$ (this is an even linear language thanks to Theorem 7.2.13). So $C_i = \{u \bullet^{|x|} v; uxv \in \mathrm{L}(G)$ is reduced using $i = (L_\ell, x \to y, L_\mathrm{r})$ into a word from $\mathrm{L}(G)\}$. This set can be transformed into the even linear language

$$C'_i = \{uxv; uxv \in \mathrm{L}(G) \text{ is reduced using } i \text{ into a word from } \mathrm{L}(G)\}$$

using Theorem 7.2.14. Let us join those sets to get $C' = \bigcup_{i\in I} C'_i$ (according to Theorem 7.2.13, the language C' is an even linear language). Thus, $C' = \{w; w \in \mathrm{L}(G)$ can be reduced by M to a word of $\mathrm{L}(G)\}$.

Now if every word of $\mathrm{L}(G)$ is either a member of C' or accepted directly, then surely $\mathrm{L}(G) \subseteq \mathrm{L}(M)$, otherwise, this inclusion obviously does not hold. We are able to select only the automata having this property. Let L_b be the regular language of words accepted by M directly. It suffices to compute $\mathrm{L}(G) \cap (C' \cup L_\mathrm{b})^\mathrm{c}$ (note that this is an even linear language according to Theorem 7.2.12 and Theorem 7.2.13) and then to decide according to the emptiness of the resulting language. If it is not empty, then it contains a word from $\mathrm{L}(G) \setminus \mathrm{L}(M)$. Otherwise, each word from $\mathrm{L}(G)$ is either reduced into a word from $\mathrm{L}(G)$ or accepted by M directly, therefore $\mathrm{L}(G) \subseteq \mathrm{L}(M)$. Let $\mathcal{M}_1 = \{M; M \in \mathcal{M}_0, \mathrm{L}(G) \subseteq \mathrm{L}(M)\}$.

Then for each $M \in \mathcal{M}_1$, we have $\mathrm{L}(M) = \mathrm{L}(G)$. Also, if for some $M \in \mathcal{M}$ it holds $\mathrm{L}(M) = \mathrm{L}(G)$, then $M \in \mathcal{M}_1$ because obviously $M \in \mathcal{M}_0$ and $\mathrm{L}(G) \subseteq \mathrm{L}(M)$. Therefore, $\mathcal{M}_1 = \{M \in \mathcal{M}; \mathrm{L}(M) = \mathrm{L}(G)\}$. Then the answer is *yes* if and only if $\mathcal{M}_1$ is not empty.

□

The above presented proof gives an algorithm for building a restarting automaton witnessing the answer to the question posed by the problem $F_{\mathrm{EL-AUX}-f,k}$. However, the algorithm iterates over the set having the (L_k^f, f, k) property that may be quite huge. So the usability of this method may be limited in practice.

There is a natural question about even linear languages: Are RRW-automata powerful enough to accept all even linear languages? Below we prove that it is not the case. To prove this, we will need the following auxiliary theorem.

Theorem 7.2.17 ([56]). *Let $X \in \{\mathsf{RR}, \mathsf{RRW}, \mathsf{RRWW}\}$. For each X-automaton M, there is an X-automaton M' and $k > 0$ such that $\mathrm{L}(M) = \mathrm{L}(M')$ and no word longer than k symbols is accepted by M' directly (i.e. without a restart).*

Now we can proceed with the theorem announced above.

Theorem 7.2.18. *Let $L = \{ww' | w \in \{0,1\}^*, |w| = |w'|, w^{\mathrm{R}} \neq w'\} \cup \{w \in \{0,1\}^*; |w|$ is an odd number$\}$. It holds:*

- *$L \in \mathsf{EL}$, and*
- *$L \in \mathcal{L}(\mathsf{RRWW}) \setminus \mathcal{L}(\mathsf{RRW})$.*

Proof. Let us prove that L is an even linear language. We will show that L can be generated by the grammar $G = (\{0,1\}, \{\mathrm{S}, \mathrm{T}\}, \mathrm{S}, P)$, where P contains the following rules:

(i) $\mathrm{S} \to a\mathrm{S}a$ for $a \in \{0,1\}$,

(ii) $\mathrm{S} \to a\mathrm{T}b$ for $a, b \in \{0,1\}, a \neq b$,

(iii) $\mathrm{S} \to a$ for $a \in \{0,1\}$,

(iv) $\mathrm{T} \to a\mathrm{T}b$ for $a, b \in \{0,1\}$,

(v) $\mathrm{T} \to \lambda$, and

(vi) $\mathrm{T} \to a$ for $a \in \{0,1\}$.

We will first prove that $L \subseteq \mathrm{L}(G)$. Let $w \in L$. Let x be the longest prefix of the left half of w (in the case of w having odd length, we do not consider the symbol in the middle to be part of either half) such that w has a suffix x^{R}. We start from the non-terminal S by using $|x|$-times the rule (i). This way we obtain $x\mathrm{S}x^{\mathrm{R}}$. Then, if only 1 symbol is missing to obtain w, we finish the derivation by the rule (iii). Otherwise, we use the rule (ii) to obtain $xa\mathrm{T}bx^{\mathrm{R}}$ for $a, b \in \{0,1\}$ such that $a \neq b$. Then the rule (iv) is used until at most 1 symbol is missing to obtain w. Then we finish the derivation by the rule (v) or (vi).

Now we will prove that $\mathrm{L}(G) \subseteq L$. There are three ways to finish a derivation of some word — using either the rule (iii), (v), or (vi). In the case of the rules (iii) and (vi), it is clear that the obtained word belongs to L because it has odd length. In the case of the rule (v), the resulting word belongs to L as well because in the derivation of that word, there must be a switch from the non-terminal symbol S to the non-terminal symbol T (i.e. the use of the rule (ii)) that introduced two different terminal symbols on the corresponding positions in the left half and in the right half of the obtained word.

Thus, L is an even linear language. It is therefore clear that $L \in \mathcal{L}(\mathsf{RRWW})$ because all context-free languages can be accepted by an RRWW-automaton [37].

Next, we will show that L can not be accepted by any RRW-automaton. For a contradiction, let us suppose that an RRW-automaton $M = (\Sigma, \Gamma, I)$ accepts L. According to Theorem 7.2.17, we can suppose that all words accepted by M directly (without a restart) are shorter than ℓ' symbols, for some constant $\ell' > 0$.

Let I_{RW} be the set of all rewriting meta-instructions of M and let $L_{\mathrm{rd}} = \bigcup_{(L_\ell, x \to y, L_\mathrm{r}) \in I_{\mathrm{RW}}} (L_\ell \cdot \{x\} \cdot L_\mathrm{r})$ be the set of all words reduced by M. This is obviously a regular language. Similarly, $L_{\mathrm{nr}} = \{0,1\}^* \setminus L_{\mathrm{rd}}$ is the regular language consisting of all words which can not be reduced by M.

Let $w_{n,k} = (0^{2k}1)^n 0^{2k} (10^{2k})^n$ for $n > 0$ and $k > 0$. Obviously, $w_{n,k} \notin L$ as $w_{n,k}$ has even length and it is of the form ww^{R} for $w = (0^{2k}1)^n 0^k$. In general, M need not to reduce long words not belonging to L. However, we will prove the following claim, where k_0 is the constant denoted as n within the pumping lemma for regular languages (Theorem 1.1.25) applied to L_{nr}.

Claim. *The automaton M can reduce the words $w_{n,k}$ for all $n > \ell'$ and $k > k_0$.*

Proof. For a contradiction, let us suppose that for some $k > k_0$ and $n > \ell'$, the word $w_{n,k}$ can not be reduced by M. Hence, $w_{n,k} \in L_{\mathrm{nr}}$ and we can pump within the prefix 0^{2k} of $w_{n,k}$. We obtain the word $w' = (0^{2k+2i}1)(0^{2k}1)^{n-1}0^{2k}(10^{2k})^n$ for some $i > 0$ also from L_{nr}. However, w' is in L as w' has even length and starts by 0^{2k+1}, but ends by 10^{2k}. This is a contradiction to the assumption that all words accepted by M directly are shorter than ℓ' symbols.

□

Thus all words $w_{n,k}$ for $n > \ell'$ and $k > k_0$ can be reduced by M. We will additionally suppose that k_0 is greater than the size of the window of M. We will further prove that those words are always reduced by a rewriting in the innermost subword 0^{2k}.

Let us suppose that the rewriting does not occur in the innermost subword 0^{2k}. As the result of the rewriting, no more than $k-1$ symbols can be removed (because k is greater than the size of the window of M). Thus, the center of the word moves by less than $k/2$ symbols to either side. Before the rewriting, there was the word 0^k before the center and 0^k following the center — in total, there was the innermost subword 0^{2k}. After the rewriting, there will surely be a pair of corresponding symbols around the center of the word that will not be the same and thus the resulting word will belong to L. This is a contradiction to the error preserving property of M. Thus, the rewriting has to occur in the innermost subword 0^{2k}.

Let us create a language similar to L_{rd} that will mark the exact places of rewritings: $L'_{\mathrm{rd}} = \bigcup_{(L_\ell, x \to y, L_\mathrm{r}) \in I_{\mathrm{RW}}} (L_\ell \cdot \{x\bullet\} \cdot L_\mathrm{r})$ where $\bullet$ is a new symbol. Again, L'_{rd} is a regular language. Let $L_1 = \{w_{n,k'_0}; n > n_0\}$ for $k'_0 = k_0 + 1$ — note that this is a regular language as well because $L_1 = \mathrm{L}((0^{2k'_0}10^{2k'_0}1)^* 0^{2k'_0}) \cap \Sigma^{\geq 2(n_0+1)(2k'_0+1)+2k'_0}$. We define a homomorphism h by $h(0) = 0, h(1) = 1$, and $h(\bullet) = \lambda$. Now let us consider $L_2 = L'_{\mathrm{rd}} \cap h^{-1}(L_1)$. We know that all words from L_1 are reduced by M. The words in L_2 contain a special mark $\bullet$ denoting where exactly rewritings of words from L_1 occur. Remember that we already proved that the rewriting changes the innermost subword 0^{2k} in all those $w_{n,k}$

and thus also the mark • occurs at the corresponding place of each word from L_2 — but note that the mark can be placed either in the innermost subword 0^{2k} or in the following one (consider e.g. the located reduction $(0010, 010 \to 10, 0)$ where the corresponding word in L_2 would be $0010010 \bullet 0$). Still, L_2 is a regular language (remember that the class of regular languages is closed under inverse homomorphism).

However, consider the homomorphism $g(0) = \lambda, g(1) = 1$, and $g(\bullet) = \bullet$ applied to L_2. We obtain the infinite language $g(L_2) \subseteq \{1^n \bullet 1^n; n > n_0\} \cup \{1^{n+1} \bullet 1^{n-1}; n > n_0\}$ that is not a regular language (use Theorem 1.1.25). This is a contradiction because the class of regular languages is closed under homomorphism.

We conclude that there is no RRW-automaton accepting L.

□

The above presented drawbacks of grammar-based methods for creating training and testing data are significant enough to avoid using them. Though there are positive results related to the even linear grammars case, we consider this class of grammars to be too restricted to represent target languages used to adequately evaluate our method. Instead, in experiments where random targets and random training and testing data are needed and where methods for inference from FPABR samples are evaluated, we use the method based on automata presented in the next subsection.

7.2.2 Automata-Based Evaluation Method

As mentioned in Subsection 7.2.1, there is a simple way to define positive and negative reductions if we have the target language represented by a restarting automaton. The reductions performed by the automaton are called positive, the others are called negative. Similarly, the positive sample words are members of the target language, the negative sample words belong to its complement (and it is straightforward to decide if a given sample word is a positive sample or a negative sample).

Initially, we intended to test our approach on benchmarks used by others. Though there are no known benchmarks used for ABR inference methods, there are still some interesting languages to use in tests. When starting from a restarting automaton, everything is clear. It is not the case when the languages are given in some other way. We often have to start with other representation of the target language — it may be given using a grammar or some expressions like $\{0^n1^n; n \geq 0\}$. Here we can transform such representations into restarting automata. If only a very limited set of benchmark languages is considered, then the transformation could be done by hand (if possible at all).

Let us suppose that we have some fixed target language and that we want to transform it into a restarting automaton that will be used to generate training data.

Of course, there can be several different restarting automata accepting the same target language and processing particular words in very different ways. For example, one of those automata can depend on auxiliary symbols while others do not use them. Or, for example, one automaton can accept the target language while not having any rewriting meta-instruction while other automata depend on them. Then the choice of a particular automaton to perform the test with (to

create the benchmark data) can significantly influence the results achieved by the inference algorithm.

While we do not know which of the restarting automata accepting the target language is "the right one", we could try to reduce the risk of introducing such a bias by finding several different restarting automata accepting the given target language and then performing the evaluation based on all of them. However, after some initial experiments, we realized that the most natural restarting automata accepting several target languages used by other researchers for benchmarks use only very simple rewriting meta-instructions where the contexts were languages containing all words over considered alphabets. Those were considered too easy for the inference method presented in this thesis. Other automata accepting those languages looked quite awkward. Therefore, we decided to use another approach based on random restarting automata.

Despite the fact that our approach is aimed towards learning languages that can be represented using S-kR-RRWW-automata, we decided to use languages represented by general RRW-automata as targets in benchmarks. Not considering auxiliary symbols is not a big issue here due to the nature of our inference method as it nearly does not distinguish them in the inference process. The resulting restarting automata returned by our algorithm will be nearly the same without distinguishing auxiliary and input symbols — the only thing dependent on that is the input alphabet set present in the resulting restarting automata. On the other hand, allowing target languages represented by restarting automata that are neither k-reversible for any k nor single poses a big challenge for our learning method.

The target restarting automata are generated at random as follows. At first, we will describe how to generate finite state automata at random. We start with two parameters — the number of states (denoted *qsn*) and the number of accepting states (denoted *fsn*).

1. a random directed graph on *qsn* nodes $\{1, \ldots, qsn\}$ is created such that all vertices have outdegree equal to 2 (they will represent transitions by 0 and 1, respectively),

2. the node 1 is considered to be the initial state,

3. *fsn*-times a node is picked at random and marked as accepting state (duplicates are allowed, i.e. one node can be picked several times), and

4. the FSA represented by the graph is reduced and returned.

To specify a restarting automaton, we have to give both its rewriting meta-instructions and accepting meta-instructions. For simplicity, we decided that there will be always exactly one accepting meta-instruction that accepts only the empty word. This decision was made based on the following ideas:

- It simplifies benchmarks — later, you will see that in order to get an FPABR sample based on a restarting automaton, we start with a simple word and iteratively extend it several times according to the meta-instructions of that restarting automaton. If we have only one starting point (i.e. the empty word), this task is simplified.

- As we learn from FPABR samples, we do not have to decide which words are accepted directly at all (because those are simply the last members of FPABR samples) and thus we are interested mainly in the process of reducing words rather than in accepting the base language.

Thus, it remains to describe the process of generating a random rewriting meta-instruction. We start with three parameters — the upper limit on the length of the subword being replaced by the generated meta-instruction (denoted *rwn*) and parameters for creating random finite state automata representing left and right contexts of the meta-instruction (denoted as above using *qsn* and *fsn*).

1. Finite state automata representing left and right contexts are generated at random using the above described method and values *qsn* and *fsn* (let the obtained FSA's be *lfsa* and *rfsa* for left and right contexts, respectively).
2. The length of the rewritten subword x is generated at random between 1 and *rwn*.
3. The length of the replacement subword y is generated at random between 0 and $|x| - 1$.
4. The rewritten word x and the replacement word y are generated at random satisfying the above specified requirements on their lengths.
5. The meta-instruction $(lfsa, x \to y, rfsa)$ is returned.

The process of generating a random restarting automaton is then straightforward. Let us call the proposed method for generating a random restarting automaton RA-RAND. We consider the input and working alphabets equal to $\{0, 1\}$ for simplicity. We start with four parameters — the number of rewriting meta-instructions (denoted *risn*) and *rwn*, *qsn*, and *fsn* having meaning as above.

1. *risn* rewriting meta-instructions (let us denote the resulting set *ris*) are generated at random using parameters *rwn*, *qsn*, and *fsn* as described above.
2. The set of accepting meta-instructions containing the single member accepting only the empty word is denoted $ais = \{(\{\lambda\}, \text{Accept})\}$.
3. The automaton $(\{0, 1\}, \{0, 1\}, ris \cup ais)$ is returned.

Having a random target, the next step is to create training and testing data sets. Initially, we simply tried to generate random words and parse them using the random target automaton. However, this approach had problems with generating positive data in the case of very sparse languages and with generating negative data in the case of very dense languages (by a *very dense language* (or a *very sparse language*) we mean a language for which it is hard to find any word outside (or from, respectively) that language by generating random words, i.e. we do not consider the definition from e.g. [25]). Therefore, we decided to use a different approach, at least in the case of positive samples. Instead of generating a random word and parsing it, we start with a directly accepted word and expand it randomly several times to obtain a FPABR sample. Remember that the only

word that is accepted directly is the empty word. Thus, the starting point of generating the FPABR sample is clear.

Another issue present in the initial approach consisted in the risk of considering target automata doing hardly any rewriting when parsing words (note we are dealing with random automata where no idea on how to process more complex input words is given by a human). For the purpose of presenting this idea, let us allow accepting meta-instructions not restricted to the empty word only. Even quite rich training data consistent with such automata could contain mainly FPABR samples where no reduction is performed. Then the inference algorithm would be inferring mainly the language contained in the accepting meta-instruction, i.e. it would do something much easier and we could be misled by surprisingly good results when such targets are used in benchmarks. Consider for example the restarting automaton $M = (\{0,1\}, \{0,1\}, I)$ where I contains the following meta-instructions:

- $(\{\lambda\}, 0 \to \lambda, \{\lambda\})$,
- $(\{\lambda\}, 1 \to \lambda, \{\lambda\})$, and
- $(\{0^i1^j0^k1^\ell; i, j, k, l \in \mathbb{N}\}, \text{Accept})$.

Obviously, only words 0 and 1 can have the ABR longer than just one step. However, both can be accepted directly as well. Let us suppose we use M to represent the target. Now we can generate random FPABR samples consistent with M to be used as training data. However, we will have only very simple FPABR samples consisting of one or two steps. If training data generated this way are supplied to the inference algorithm, we will be rather inferring the base language. To avoid this problem, we again benefit from the idea of only the empty word being accepted directly. Then we can be sure the generated training samples contain at most one FPABR sample consisting of only one step, i.e. the ABR for the empty word. This approach surely increases the risk of obtaining an automaton accepting only a very limited number of words, maybe the only one being the empty word. If the obtained automaton can not be used to generate enough training and testing samples (the considered threshold will be specified later), we will consider it to be singular and we will scratch it and then we will try a new one.

To recap — to obtain FPABR samples, we start with the single word accepted directly (the empty word) and expand it iteratively several times according to the rewriting meta-instructions of the target automaton. At each step, we assign an equal probability to choosing any of the possible words that can be directly reduced to the current one and also to the possibility of closing this ABR and returning the current sequence as the generated FPABR sample. A random choice is performed and we continue accordingly.

One run of an experiment is performed as follows. There are several input parameters:

positiveTrainingSamplesN (an integer) The number of requested (positive) training samples (i.e. FPABR samples).

testingSamplesN (an integer) The number of required positive and negative testing sample words (i.e. *testingSamplesN* of positive sample words and also *testingSamplesN* of negative sample words will be used).

positiveSamplesCandidatesN (an integer) We generate positive training and testing samples using the above described method (though in the case of testing samples, only the most extended words of FPABR samples will be used). It may seem that it would be enough to randomly generate *positiveTrainingSamplesN*+*testingSamplesN* different FPABR samples. But note that for example one training sample (i.e. one sample of the ABR) can contain some testing sample (i.e. a word) as one of the words present in that ABR. We do not want such things to occur — we want to test the resulting automata using testing data completely free of words present in the training samples. It is more interesting to see the performance on unseen data as we can easily extend any inference algorithm by using a memory to remember all training samples to perform on them perfectly. Moreover, imagine the situation when a benchmark uses for some reason testing samples sets that share many samples with corresponding training sets. It could easily hide the difference between inference algorithms compared using that benchmark.

Therefore, some modification of the simple approach is needed. At first, we generate *positiveSamplesCandidatesN* FPABR samples at random. They will be then used as a source of samples from which to build the required training and testing samples sets. If there will not be enough samples for us to satisfy other input requirements (e.g. that testing samples are free of words present in the training samples), we scratch the current experiment and proceed to the next one. Details will be described below.

Please note that a testing sample is just a word, not a whole FPABR sample. The correct ABR of this sample word can contain words present in the positive training samples set, only the testing sample itself can not occur in the training set!

negativeSamplesCandidatesN (an integer) Creating *testingSamplesN* negative samples consists in generating words at random, removing duplicates, and filtering out positive samples. But note that the target language may be very dense and it may be quite hard to get a word outside the language at random. Or there may be no more than $testingSamplesN - 1$ possible negative samples at all. Therefore, we adopt an approach where *negativeSamplesCandidatesN* sample words are generated at random (containing possibly both positive and negative samples and also duplicates) and then we try to obtain the negative samples we are interested in from this data. If there are not enough samples satisfying our requirements, we scratch the experiment and proceed with another one.

positiveSamplesMaxExpansionSteps (an integer) The maximum number of expansion steps to perform when generating an FPABR sample.

negativeTestingSamplesMaxWordLength (an integer) The maximum possible length of a negative sample word.

randomTargetSource (an algorithm) The source of random target automata (in this thesis, the method RA-RAND, described in Subsection 7.2.2, will be supplied).

The process of performing an experiment with an inference algorithm A is then as follows (the algorithm several times selects given number of samples from some pool — for improved readability, exceptional cases are not considered below but here instead: if there is not enough samples to obtain the required number of training samples or either positive or negative testing samples, then the experiment is scratched immediately as described above for parameters *positiveSamplesCandidatesN* and *negativeSamplesCandidatesN*):

1. A random target restarting automaton is obtained using the supplied algorithm *randomTargetSource* (i.e. some method for creating random automata, e.g. RA-RAND).

2. *positiveSamplesCandidatesN* FPABR samples (each sample having at most *positiveSamplesMaxExpansionSteps* expansion steps) are generated at random and then duplicates are removed.

3. *positiveTrainingSamplesN* samples are extracted from the just created set of samples to form the training data set.

4. The remaining generated samples are simplified so that only the most extended word of each of the FPABR samples remains, duplicates are removed again, words present in the positive samples set are removed as well, and finally randomly selected *testingSamplesN* of the remaining words form the final testing positive samples.

5. *negativeSamplesCandidatesN* random words having the length less or equal to *negativeTestingSamplesMaxWordLength* are generated (at first lengths are generated randomly and then random words having those lengths are generated), then duplicates are removed, words accepted by the target automaton are removed as well, and finally *testingSamplesN* words are used as testing negative word samples.

6. A hypothesis is inferred from (positive) training data using the supplied algorithm A.

7. The performance of the hypothesis on positive and negative testing data is measured (i.e. the ratio of correctly classified positive samples and the ratio of correctly classified negative samples are computed).

Note that even if there are many target automata generated by the benchmark, it can be the case that only a very small fraction of them will be actually used to perform a run of the algorithm A being evaluated. It depends on the considered parameters of the benchmark. Let us consider e.g. the language $\{0^n1^n; n > 0\}$. If *positiveSamplesMaxExpansionSteps* $= 10$, then the number of generated unique FPABR samples will be very small. Moreover, consider that we do not allow testing samples to be present in training samples. This limits the number of training and testing samples we can obtain at all. For other target automata, it

can be hard to find any negative samples. As you can see, it is important to set the parameters carefully and even to consider whether to use this benchmark at all, particularly if you are interested in very sparse or dense target languages.

We consider the presented benchmark to be appropriate to reasonably evaluate the performance of ABR inference algorithms as it allows to perform many tests using random targets. In Section 7.3 we will use this approach as well as the simple approach based on fixed target languages and fixed training samples to evaluate our method.

7.3 Results

In Chapter 6 we proposed a new method for solving a not so common learning task — instead of traditional inference of a language from sample words, we infer a model of the ABR from FPABR samples. Each inferred model of the ABR defines a language. In the following benchmarks, we will not evaluate the learned way to perform the ABR, but only the language represented by the inferred model of the ABR. This section evaluates our method on two different benchmarks. First, we will show that our method is able to learn interesting languages in Subsection 7.3.1 using a fixed set of target languages as well as fixed training samples. The experiments presented there can also be seen as a comparison with other methods (particularly with LARS [21]). However, the general performance of our method needs to be analyzed more thoroughly. In Subsection 7.3.2 we present results achieved in benchmarks working with random targets as proposed in Subsection 7.2.2.

7.3.1 Simple Benchmark

The goal of this subsection is to show that our method is able to infer some basic languages (at least when suitable samples are supplied). Eyraud, Higuera, and Janodet [21] proposed the LARS method for learning rewriting systems. These systems are close to restarting automata. They used a set of languages that are easily understandable for humans and they evaluated the performance of their method on inferring those languages from given samples. Because of the similarity of inferring rewriting systems and inferring restarting automata, we decided to perform tests on similar languages too.

Our method expects FPABR samples. The input considered in LARS is not so rich regarding the positive samples. On the other hand, they work with negative samples in contrast to our method. To obtain the required input in our benchmark, we decided to choose manually some training samples that seem quite exemplary for a human. Note that this makes the comparison between LARS and our method harder. However, remember that the goal of this subsection is to show the ability to infer interesting languages. The results presented here were published in [31]. They were obtained by the Omega2 method (a method for learning S-ZR-RRWW-automata). It is important to have this in mind as it particularly shows that we are able to learn even the least capable version of our model to accept quite complex languages that other researchers consider in their benchmarks as well. Below we also present the training samples used in our experiments — for convenient reading, we underline parts of words that can be

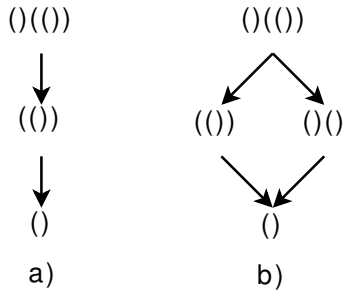


Figure 7.2: Different ABR's of the word ()(()) performed by two restarting automata inferred by Omega2 method. An arrow from u to v means that u can be directly reduced to v. The first automaton performs the ABR depicted under a). The second automaton performs the two ABR's depicted under b).

rewritten to perform particular reductions. Note however, that exact locations of rewritings were not supplied to the learning algorithm. Also, there can be several possible places for a rewriting but we denote only one of them. As you will see, the FPABR samples often do not end with the empty word. Our inference method does not restrict the expected training samples in this way. The empty word as the last member of a FPABR sample is considered in the RA-RAND method for generating restarting automata at random. As we already said, this subsection deals with manually selected input samples. Therefore, we can use samples that are suitable for the languages used in [21].

At first, Dyck language L_1 (words of balanced parentheses) was used. It is a common example of a context-free language. The algorithm LARS learns the correct grammar from about 20 words. We were able to learn an automaton accepting this language from the following FPABR sample: $(\underline{()}())() \Rightarrow (\underline{()})() \Rightarrow \underline{()}() \Rightarrow ()$. Obviously, this is a very simple example, but note that, in addition to the words themselves, a relation on them (i.e. the way of performing the ABR) was given to the learning algorithm.

We will use the Dyck language also to show that using other input samples, the same language can be obtained but the inferred ABR will be different. This happens when the input consists of the following FPABR sample: $(()())()((\underline{()})(())) \Rightarrow (\underline{()}())()(()(())) \Rightarrow (())()(()(\underline{()})) \Rightarrow (())()(\underline{()}()) \Rightarrow (\underline{()})()(()) \Rightarrow \underline{()}()(()) \Rightarrow ()(\underline{()}) \Rightarrow \underline{()}() \Rightarrow ()$. In the former case, the inferred restarting automaton always removes the left-most subword (), whereas in the latter, it learnt to remove any subword (). Thus, the two inferred automata perform different ABR's. This can be seen in Fig. 7.2, where both automata parse the word ()(()). The computation of the former automaton depicted under a) is deterministic, the latter automaton is non-deterministic. The two its computations on the word ()(()) are depicted under b).

The language $L_2 = \{0^n1^n; n > 0\}$ is a common example of a language that is not regular. We were able to learn an automaton accepting L_2 from the sample $00\underline{01}11 \Rightarrow 0\underline{01}1 \Rightarrow 01$. A similar language $L_3 = \{0^n1^n2^m; n, m > 0\}$ was inferred from the following two samples:

- $00\underline{01}1122 \Rightarrow 0\underline{01}122 \Rightarrow 01\underline{22} \Rightarrow 012$, and

- $0\underline{01}1222 \Rightarrow 01\underline{2}22 \Rightarrow 01\underline{2}2 \Rightarrow 012$.

Another alteration gives the language $L_4 = \{0^m1^n; 0 < m \leq n\}$, that was learnt from the sample $00\underline{01}1111 \Rightarrow 0\underline{01}111 \Rightarrow 0\underline{1}11 \Rightarrow 0\underline{1}1 \Rightarrow 01$. To compare, LARS was able to learn these languages from about 20 samples.

We were also able to learn the language $L_5 = \{0^m1^n; m \neq n\}$. It was inferred from the following two samples:

- $00\underline{01}1111 \Rightarrow 0\underline{01}1111 \Rightarrow \underline{01}111 \Rightarrow \underline{1}11 \Rightarrow \underline{1}1 \Rightarrow 1$, and
- $00000\underline{01}11 \Rightarrow 0000\underline{01}1 \Rightarrow 0000\underline{1} \Rightarrow \underline{0}00 \Rightarrow \underline{0}0 \Rightarrow 0$.

Again, LARS inferred the language from about 20 samples.

Next, we tried the language $L_6 = \{w \in \{0,1\}^*; |w|_0 = |w|_1\}$. We inferred an automaton accepting this language from the following samples:

- $00\underline{01}11 \Rightarrow 0\underline{01}1 \Rightarrow \underline{01} \Rightarrow \lambda$,
- $11\underline{10}00 \Rightarrow 1\underline{10}0 \Rightarrow \underline{10} \Rightarrow \lambda$,
- $0\underline{01}10101 \Rightarrow \underline{01}0101 \Rightarrow \underline{01}01 \Rightarrow \underline{01} \Rightarrow \lambda$, and
- $1\underline{10}01010 \Rightarrow \underline{10}1010 \Rightarrow \underline{10}10 \Rightarrow \underline{10} \Rightarrow \lambda$.

An automaton for a similar language $L_7 = \{w \in \{0,1\}^*; 2 \cdot |w|_0 = |w|_1\}$ was inferred from the following samples:

- $\underline{011}011 \Rightarrow \underline{011} \Rightarrow \lambda$,
- $0\underline{01}111 \Rightarrow \underline{011} \Rightarrow \lambda$,
- $10\underline{011}1 \Rightarrow \underline{101} \Rightarrow \lambda$,
- $\underline{110}011 \Rightarrow \underline{011} \Rightarrow \lambda$,
- $1\underline{11}010 \Rightarrow \underline{110} \Rightarrow \lambda$, and
- $\underline{110}110110 \Rightarrow \underline{110}110 \Rightarrow \underline{110} \Rightarrow \lambda$.

The algorithm LARS was able to infer both L_6 and L_7 from about 30 samples.

The next positive result was obtained for the language of Lukasiewicz, here given using a grammar: $L_8 = L((\{0,1\}, \{S\}, S, \{S \to 1; S \to 0SS\}))$. An automaton accepting this language was learnt from the ABR of just one word! It was $0\underline{01}1011 \Rightarrow \underline{01}011 \Rightarrow \underline{011} \Rightarrow 1$.

The language $L_9 = \{0^m1^m2^n3^n; m, n \geq 0\}$ can be used to show an important negative result. Though LARS was able to infer this language, our approach was not able to learn from our given samples. We conjecture that it can not be learnt using our approach using any input samples. Our model is powerful enough to accept this language, for example using the automaton $(\{0,1,2,3\}, \{0,1,2,3\}, \{(0^*, 01 \to \lambda, (1{+}2{+}3)^*), (2^*, 23 \to \lambda, 3^*), (\{\lambda\}, \text{Accept})\})$. This automaton describes our feeling of what the right ABR for this language is. However, it is not possible to infer any automaton accepting this language using FPABR samples corresponding to the given automaton. It can be seen as

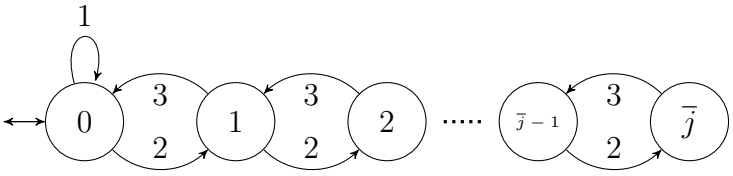


Figure 7.3: A zero-reversible automaton B.

follows. Let us have a set of input FPABR samples. Let us consider all compatible reductions performing the rewrite $01 \to \lambda$. Let T denote the set of right contexts of these reductions (they are of the form $1^i2^j3^j$ for $i, j \geq 0$). Let $\overline{j}$ denote the maximum value of j used in the input samples. How will the corresponding inferred rewriting meta-instruction look like? Let us focus on its right context. Let A denote the zero-reversible language inferred from T by the ZR algorithm. The automaton B in Fig. 7.3 is zero-reversible and accepts all words from T. As the ZR algorithm returns the smallest zero-reversible language containing T, the automaton A accepts a subset of $L(B)$ only. Hence, for example $1^32^{j+1}3^{j+1}$ will be rejected by A and the word $w = 0^41^42^{j+1}3^{j+1}$ will neither be reduced nor accepted by the automaton M returned by our algorithm (note that neither the 23 will be removed from the word as the left context of the corresponding meta-instruction will be a subset of $\{2\}^*$). Finally, as $w \in L_9$ is rejected by M, we obtain $\mathrm{L}(M) \neq L_9$.

The language $L_{10} = \{w \in \{0,1\}^*; w = w^{\mathrm{R}}\}$ of palindromes can be used to show the use of auxiliary symbols (here it will be X; the FPABR samples used in previous examples contained no auxiliary symbols). We were able to learn this language from less than 25 FPABR samples (LARS was unable to learn this language at all). The samples look like $00\underline{11}00 \Rightarrow 0\underline{0X0}0 \Rightarrow \underline{0X0} \Rightarrow \underline{X} \Rightarrow \lambda$. Note the symbol X inserted to mark the center of the word. You can see all input samples below:

- $0\underline{00}0 \Rightarrow \underline{0X0} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $1\underline{00}1 \Rightarrow \underline{1X1} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $00\underline{00}00 \Rightarrow 0\underline{0X0}0 \Rightarrow \underline{0X0} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $11\underline{00}11 \Rightarrow 11\underline{X11} \Rightarrow \underline{1X1} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $\underline{000} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $0\underline{000}0 \Rightarrow \underline{0X0} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $1\underline{000}1 \Rightarrow \underline{1X1} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $\underline{010} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $0\underline{010}0 \Rightarrow \underline{0X0} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $1\underline{010}1 \Rightarrow \underline{1X1} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $1\underline{11}1 \Rightarrow \underline{1X1} \Rightarrow \underline{X} \Rightarrow \lambda$,

- $0\underline{11}0 \Rightarrow \underline{0X0} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $11\underline{11}1 \Rightarrow 1\underline{1X1}1 \Rightarrow \underline{1X1} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $00\underline{11}00 \Rightarrow 0\underline{0X0}0 \Rightarrow \underline{0X0} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $\underline{111} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $1\underline{111} \Rightarrow \underline{1X1} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $0\underline{111}0 \Rightarrow \underline{0X0} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $\underline{101} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $1\underline{101}1 \Rightarrow \underline{1X1} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $0\underline{101}0 \Rightarrow \underline{0X0} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $01\underline{11}10 \Rightarrow 0\underline{1X1}0 \Rightarrow \underline{0X0} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $00\underline{11}100 \Rightarrow 00\underline{1X1}00 \Rightarrow 0\underline{0X0}0 \Rightarrow \underline{0X0} \Rightarrow \underline{X} \Rightarrow \lambda$,
- $10\underline{00}01 \Rightarrow 1\underline{0X0}1 \Rightarrow \underline{1X1} \Rightarrow \underline{X} \Rightarrow \lambda$, and
- $110\underline{00}011 \Rightarrow 11\underline{0X0}11 \Rightarrow 1\underline{1X1}1 \Rightarrow \underline{1X1} \Rightarrow \underline{X} \Rightarrow \lambda$.

We tried also another language not considered by [21] to show that even the ABR of a language outside of the class of context-free languages can be learnt. We used the language $L_\alpha = \{w = (011)^i(01)^j; i, j \geq 0 \text{ and } (\exists k)(|w|_1 = 2^k)\} \cup \{w = (01)^i(011)^j; i, j \geq 0 \text{ and } (\exists k)(|w|_0 = 2^k)\}$ from [38]. This language is not context-free as $L_\alpha \cap \{(01)^n; n \geq 0\} = \{(01)^{2^n}; n \geq 0\}$. We presented only the FPABR sample $01\underline{0}10101 \Rightarrow 0110\underline{1}01 \Rightarrow 0\underline{1}1011 \Rightarrow 010\underline{1}1 \Rightarrow 01\underline{0}1 \Rightarrow 0\underline{1}1 \Rightarrow 01$. Even from this single sample, we infer an automaton performing the ABR of L_α despite the complexity of the language itself.

7.3.2 Benchmarks Using Random Targets

The results presented in Subsection 7.3.1 can be suspected from being good because of manually selecting the right input samples. Moreover, the considered target languages can be seen as too simple — remember we were able to learn most of them using even the simplest version of our model. To obtain more significant results, we use the benchmark proposed in 7.2.2. The benchmark can be finely tuned to measure performance under different conditions. We evaluate our Omega* method for inferring S-kR-RRWW-automata for all $k \in \{0, 1, 2, 3\}$ considering the following benchmark settings:

positiveSamplesCandidatesN We considered 3000 positive sample candidates (note that this limit contains both the training and testing samples).

negativeSamplesCandidatesN We considered 1000 negative samples candidates.

positiveSamplesMaxExpansionSteps We considered the value 10.

negativeTestingSamplesMaxWordLength We considered the value 20.

positiveTrainingSamplesN We considered three different values to see the improving performance when more data is supplied — the values were 20, 40, and 80.

testingSamplesN We considered 100 positive testing samples and 100 negative testing samples.

randomTargetSource Random target restarting automata were generated using the method RA-RAND, described in Subsection 7.2.2, where we used the following parameters:

(qsn,fsn) The pairs containing the number of states and the number of accepting states for finite state automata contained within meta-instructions of restarting automata were $(4, 2)$ and $(8, 4)$.

(risn,rwn) The pairs containing the number of rewriting meta-instructions and the length limit for the rewritten subword were $(2, 1), (4, 1), (4, 2), (8, 2)$, and $(12, 2)$.

For each set of benchmark parameters, 100 runs of the experiment were performed — 100 experiments inferring S-ZR-RRW-automata, 100 experiments inferring S-1R-RRW-automata and so on. Experiments inferring S-kR-RRW-automata considering different values of k were performed using *the same* data (even the generated target automaton was the same). Also, experiments considering less than 80 training samples are nearly the same as those considering 80 samples — only the training set was restricted to contain exactly the required number of samples.

In total, 12000 experiments were performed. The output of each experiment is a pair (pOK, nOK) where pOK is the ratio of correctly classified positive test samples (also called *sensitivity*) and nOK is the ratio of correctly classified negative test samples (also called *specificity*). Moreover, we present also the value of $pOK * nOK$ as a combined value of both pOK and nOK, and we denote it as *prod*. In addition to the mean values, we also estimate the standard deviation based on a sample. We write the values in form of $(pOK \pm pOK_{\text{stdev}}, nOK \pm nOK_{\text{stdev}}, prod \pm prod_{\text{stdev}})$ where the values $pOK_{\text{stdev}}, nOK_{\text{stdev}}$, and $prod_{\text{stdev}}$ are equal to the above mentioned standard deviation of pOK, nOK, and *prod*, respectively.

The average performance (expressed as defined above) over all 12000 experiments was $(0.87 \pm 0.25, 0.64 \pm 0.42, 0.52 \pm 0.40)$. At first glance, this performance is not very good because of the high standard deviations. But note that the class of automata used to represent target languages is much richer compared to the class of models we can infer.

Moreover, the combined value $prod = 0.52$ means that, on average, the performance on both negative and positive testing data does not go below the value 0.52. And even when the ratio on either positive or negative testing data equals 0.52, then the performance on the other testing data equals 1.00. That would e.g. mean a perfect performance on positive testing data and slightly above one half of correctly classified negative testing data.

number of training samples: 20			
reversibility	*pOK*	*nOK*	*prod*
0	0.99 ± 0.05	0.13 ± 0.28	0.12 ± 0.26
1	0.94 ± 0.13	0.76 ± 0.35	0.70 ± 0.34
2	0.68 ± 0.27	0.94 ± 0.12	0.63 ± 0.26
3	0.26 ± 0.20	0.98 ± 0.04	0.25 ± 0.19

number of training samples: 40			
reversibility	*pOK*	*nOK*	*prod*
0	1.00 ± 0.02	0.09 ± 0.23	0.09 ± 0.23
1	0.99 ± 0.04	0.68 ± 0.39	0.68 ± 0.39
2	0.95 ± 0.09	0.83 ± 0.26	0.79 ± 0.27
3	0.73 ± 0.20	0.94 ± 0.10	0.69 ± 0.20

number of training samples: 80			
reversibility	*pOK*	*nOK*	*prod*
0	1.00 ± 0.00	0.07 ± 0.21	0.07 ± 0.21
1	1.00 ± 0.01	0.66 ± 0.40	0.66 ± 0.40
2	0.99 ± 0.02	0.76 ± 0.34	0.75 ± 0.34
3	0.96 ± 0.06	0.87 ± 0.20	0.84 ± 0.20

Table 7.1: Results achieved by Omega* for different levels of reversibility and different sizes of training sets.

Other interesting point to see is when $prod = 0.52$ and both pOK and nOK are equal to its square root, i.e. $pOK = nOK = 0.72$. Thus, a great portion of testing samples was identified correctly.

We considered a vast set of possible benchmark configurations — some of them were not even expected to show some good results but they were included to obtain a complete set of results. On the other hand, some of the results were surprisingly good. From this point of view, the achieved overall results seem interesting.

Because of the number of performed experiments, we do not analyze all of the results here. You can see the results of all benchmarks in the file named experiments.ods on the enclosed CD.

We consider it interesting to see how the performance varies when changing particularly the considered reversibility level in the inference algorithm and the size of the training samples set. Those results are given in Tab. 7.1.

In Figures 7.4, 7.5, and 7.6, you can see boxplots showing the performance (the *prod* value is considered there) for different reversibility levels when 20, 40, and 80 training samples were supplied, respectively. The boxes in the plots contain elements as follows. A median is denoted by a long red horizontal line. Edges of the boxes represent the 25th and the 75th percentiles. Whiskers extend to the ultimate data points not considered as outliers that are denoted individually by short red lines. Means and standard deviations are summarized in Tab. 7.1.

In the case of 20 training samples supplied to the learning algorithm, we see quite poor performance when only the reversibility level equal to 0 is considered. We conjecture the problem here is that even small number of training samples

leads to over-generalization when learning contexts used in meta-instructions. Then the performance improves as we consider the reversibility level equal to 1. However, higher levels of reversibility lead to decreasing performance. We conjecture that this is because 20 training samples do not provide enough information to infer suitable but more complex languages used in meta-instructions.

In the case of 40 training samples, the poor performance when the reversibility level is equal to 0 is not surprising. If over-generalization was the problem when less samples were supplied, then now the same inevitably happened. The performance for the reversibility level equal to 1 is slightly worse than before. We expected this to occur because more positive samples forced the algorithm to generalize more than before and that led to accepting more negative samples as our model considering 1-reversible languages in contexts was not powerful enough to represent the right target language. However, the performance in the case of the reversibility level equal to 2 and 3 improved dramatically.

When we double the number of training samples supplied to the learning algorithm once more, the performance for the reversibility level equal to 1 is again only slightly worse than before. The performance for the reversibility level equal to 2 is also slightly worse. The need for higher reversibility levels in the case of richer training samples sets is obvious. On the other hand, the performance for the reversibility level equal to 3 is much better. In fact, it is by far the best result achieved in our benchmark! The achieved *prod* value was 0.84 — note that the square root of this value is 0.92. Thus, if either *pOK* or *nOK* goes below this value (and we know $prod = 0.84$), then the second goes above, but the minimum is 0.84 (and then the other ratio is equal to 1.00).

To give you an idea of how input and output of our method look like, we present some examples below. At first, let us look on one of the poor runs. The unknown target language was the language accepted by the restarting automaton M_{target} having the following meta-instructions (we selected one of the most simple benchmarks — in the sense that it is easy to describe it):

- $((0+1)^*, 0 \to \lambda, ((0+1)(0+1))^*)$,
- $(0+(1+(00+01))(01+10+11+000+001)^*(1+00), 1 \to \lambda, (1+00+01)^*)$, and
- $(\{\lambda\}, \text{Accept})$.

The training data set consisted of the following FPABR samples. The underlined symbols denote the symbols that can be deleted in order to obtain the following word. However, it does not necessarily correspond to the symbol deleted by restarting automaton M_{target} representing the target language. But note that exact locations of rewritings were not supplied to the learning algorithm. Actually, the underlined symbols correspond to the first-opportunity rewritings used by our learning algorithm.

- $0001001 \Rightarrow 000100 \Rightarrow 00010 \Rightarrow 0010 \Rightarrow 010 \Rightarrow 01 \Rightarrow 0 \Rightarrow \lambda$,
- λ,
- $00 \Rightarrow 0 \Rightarrow \lambda$,

- $\underline{0}000110010 \Rightarrow \underline{0}00110010 \Rightarrow 0011\underline{0}010 \Rightarrow 0011010\underline{0} \Rightarrow 00\underline{1}101 \Rightarrow \underline{0}0101 \Rightarrow 0\underline{1}01 \Rightarrow \underline{0}01 \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$,
- $0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$,
- $\underline{0}0100010 \Rightarrow 01\underline{0}0010 \Rightarrow 01001\underline{0} \Rightarrow 01\underline{0}01 \Rightarrow 0\underline{1}01 \Rightarrow \underline{0}01 \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$,
- $01\underline{0}0 \Rightarrow 01\underline{0} \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$,
- $\underline{0} \Rightarrow \lambda$,
- $\underline{0}011 \Rightarrow 0\underline{1}1 \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$,
- $\underline{0}0000 \Rightarrow \underline{0}000 \Rightarrow \underline{0}00 \Rightarrow \underline{0}0 \Rightarrow \underline{0} \Rightarrow \lambda$,
- $00001\underline{0}0 \Rightarrow \underline{0}000010 \Rightarrow 00001\underline{0} \Rightarrow 0000\underline{1} \Rightarrow \underline{0}000 \Rightarrow \underline{0}00 \Rightarrow \underline{0}0 \Rightarrow \underline{0} \Rightarrow \lambda$,
- $0011\underline{0}00000 \Rightarrow 0011\underline{0}0000 \Rightarrow 0011\underline{0}000 \Rightarrow 0011\underline{0}00 \Rightarrow 00\underline{1}100 \Rightarrow 001\underline{0}0 \Rightarrow 001\underline{0} \Rightarrow \underline{0}01 \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$,
- $\underline{0}011100 \Rightarrow 0\underline{1}1100 \Rightarrow 0\underline{1}100 \Rightarrow 0\underline{1}00 \Rightarrow \underline{0}00 \Rightarrow \underline{0}0 \Rightarrow \underline{0} \Rightarrow \lambda$,
- $\underline{0}001 \Rightarrow \underline{0}01 \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$,
- $0\underline{1}110001 \Rightarrow 0110001 \Rightarrow 011\underline{0}00 \Rightarrow 011\underline{0}0 \Rightarrow 011\underline{0} \Rightarrow 0\underline{1}1 \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$,
- $0100\underline{1}100 \Rightarrow 0100\underline{1}0 \Rightarrow 01\underline{0}010 \Rightarrow 01\underline{0}10 \Rightarrow 011\underline{0} \Rightarrow 0\underline{1}1 \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$,
- $01\underline{0} \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$,
- $011\underline{0} \Rightarrow 0\underline{1}1 \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$,
- $\underline{0}01001 \Rightarrow 0100\underline{1} \Rightarrow 01\underline{0}0 \Rightarrow 01\underline{0} \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$,
- $0000011001 \Rightarrow 00000\underline{1}100 \Rightarrow 00000\underline{1}0\underline{0} \Rightarrow \underline{0}000010 \Rightarrow \underline{0}00010 \Rightarrow \underline{0}0010 \Rightarrow 001\underline{0} \Rightarrow \underline{0}01 \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$.

On the presented training samples, our algorithm returned the restarting automaton M_0 having the following meta-instructions (considering the reversibility level equal to 0):

- $((0+1)^*, 0 \to \lambda, (0+1)^*)$,
- $((0+1)^*, 1 \to \lambda, (0+1)^*)$, and
- $(\{\lambda\}, \text{Accept})$.

This restarting automaton accepts all words over the alphabet $\{0,1\}$. Thus, it is not surprising that its performance was $pOK = 1.00$ and $nOK = 0.00$. On the other hand, in the case of the reversibility level equal to 1, the inferred restarting automaton M_1 contained the following meta-instructions (note that the same set of positive training samples was used):

- $(\lambda + 00^*1(0^*1)^*, 0 \to \lambda, (0+1)^*)$,
- $(0(1^*0)^*, 1 \to \lambda, (1 + 0(00)^*1)^*(00)^*)$, and

- $(\{\lambda\}, \text{Accept})$.

The performance on positive samples could not even be better than before ($pOK = 1.00$), but here the resulting automaton performed perfectly also on the negative testing data with $nOK = 1.00$ as well. The resulting automaton did not reduce any of the supplied negative testing samples. Let us see its computation on one particular positive testing sample (we underline symbols being deleted in individual reductions):

- $0111\underline{0}0100 \Rightarrow 0111\underline{0}0100 \Rightarrow 0111\underline{0}100 \Rightarrow 0111\underline{0}0 \Rightarrow 01111\underline{0} \Rightarrow 0\underline{1}111 \Rightarrow 0\underline{1}111 \Rightarrow 0\underline{1}1 \Rightarrow 0\underline{1} \Rightarrow \underline{0} \Rightarrow \lambda$.

Note that, despite its quite complex description, M_1 simply accepts the language $\mathrm{L}(M_1) = \{w \in \{0,1\}^*; w \text{ starts with 0 or is equal to } \lambda\}$. It can be seen as follows. Let us have a word w equal to λ or starting with 0. We will prove by induction on the length of w that M_1 accepts w.

- If $w = \lambda$, it is obviously accepted.
- If $w = 0$, it is reduced to λ and thus accepted.
- If $w = 0^i w'$ for some $i > 1$ and $w' \in \{0,1\}^*$, then using the first meta-instruction, we remove all but one symbol 0 at the beginning and thus we obtain a shorter word starting with 0.
- If $w = 01^i 0w'$ for some $i > 0$ and $w' \in \{0,1\}^*$, then the second occurrence of the symbol 0 is removed (by the first meta-instruction) and we obtain a shorter word starting with 0.
- If $w = 01^i$ for some $i > 0$, we iteratively remove all 1's (using the second meta-instruction) and obtain the word consisting of only 0.

Thus, in every case, w is accepted by M_1. On the other hand, if w starts with 1, it is neither accepted directly nor reduced to any word, and thus it is not accepted by M_1.

As you can see, the performance greatly depends on both the number of training samples and the reversibility level. However, when both of them are high enough, very good results can be achieved.

7.3.3 Comments on Benchmarks

Our method for learning languages is inspired by an approach humans can use to parse sentences. The complex task of learning a language is simplified by learning the ABR from FPABR samples. In our benchmark presented in Subsection 7.3.1, usually just small training sets sufficed. Also, the method was able to infer a non-context-free language. The results presented in Subsection 7.3.2 show that even in the case of random and quite complex languages, a great portion of target languages can be learnt. Therefore, we consider our method to perform really well. The results also confirmed the expected dependency of the quality of the inferred models on both the number of training samples and the considered reversibility

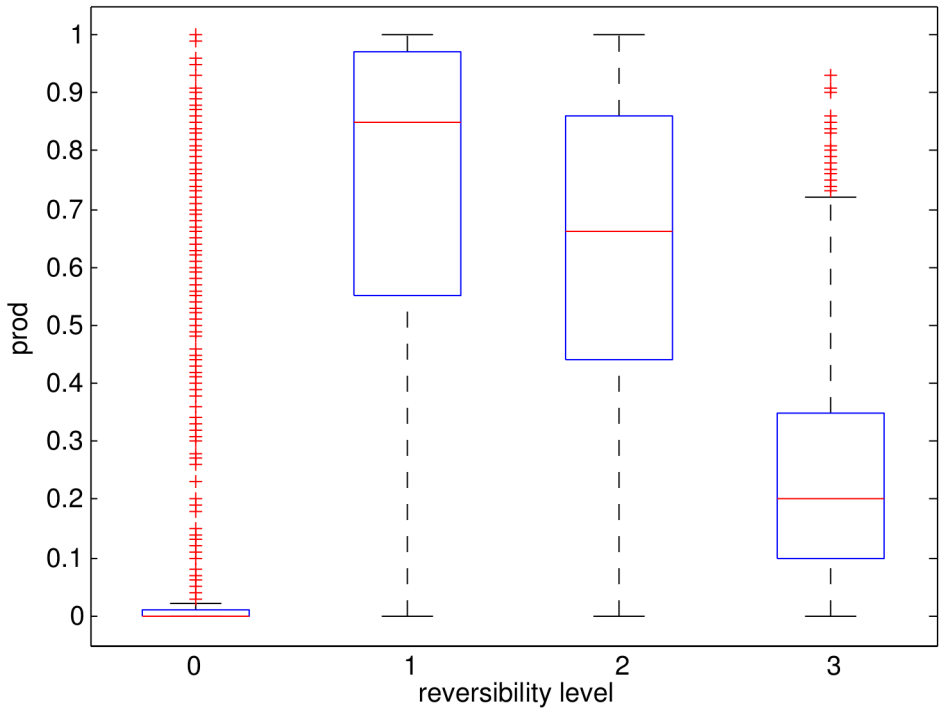


Figure 7.4: Performance for different reversibility levels when 20 training samples were supplied.

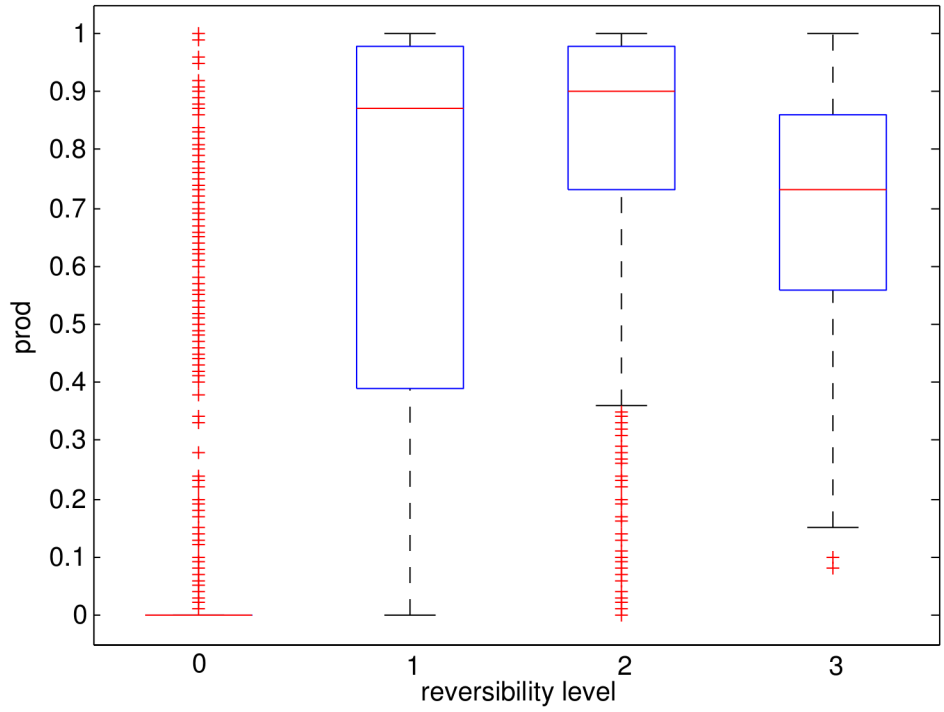


Figure 7.5: Performance for different reversibility levels when 40 training samples were supplied.

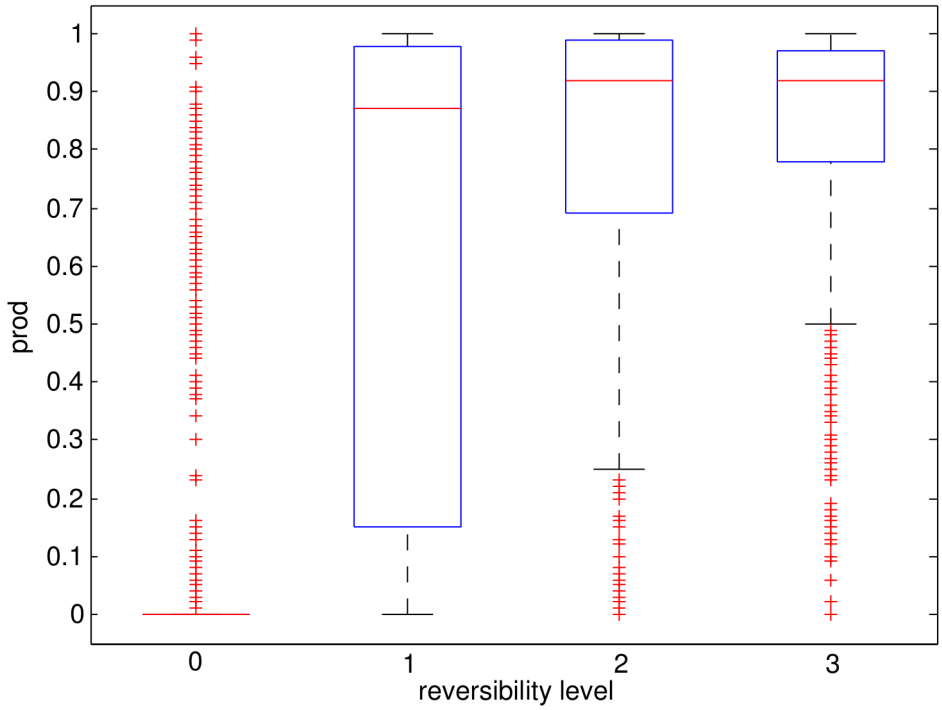


Figure 7.6: Performance for different reversibility levels when 80 training samples were supplied.

level of the models. This gives a hope for improving results in practice by tuning the parameters of the learning method.

On the other hand, the benchmarks presented in this chapter measured only the quality of the learned language, not the quality of the inferred ABR. We have used such benchmarks for easy comparison of the results with other methods of grammatical inference. Benchmarking the inference of ABR will require to design new tests.

Conclusion

In Chapter 3 we set several goals of this thesis. Let us review their current status:

1. **Complexity** We defined several versions of the problem of inferring the ABR from samples in Chapter 4. The complexity of the inference problem was analyzed in Chapter 5. The case of learning from positive and negative samples was proven to be NP-hard. Though the problem of learning from positive samples only was shown to be effectively solvable, the usefulness of the proposed method in practice is questionable. This led to the idea of looking for a new approach to the learning process.

2. **Algorithm** As indicated at the end of the previous item, a new approach to the learning process was proposed. Particularly, we relaxed our requirements by allowing any restarting automaton consistent with input samples to be returned by an inference algorithm (i.e. not necessarily the smallest one). Moreover, we considered richer input samples supplied to the algorithm (instead of arbitrary sample words and sample reductions, we expect fully positive analysis by reduction samples). For this scenario, a new method called Omega2 was proposed and later extended to Omega* method. Moreover, the new model of the ABR called S-kR-RRWW-automata was proposed and its power was analyzed. Particularly, the model has been shown to have enough power to represent all languages from GCSL. In addition, we proved that there is a hierarchy of S-kR-RRW-automata based on the value of the considered reversibility level k.

3. **Benchmark** To evaluate inference algorithms, often only very easy benchmarks are used [68, 18, 21, 31, 32]. We used one such benchmark just to show that some well known languages can be learned using our approach. It also allowed to compare our method with a slightly similar one called LARS [21]. The results were really good, almost all considered languages were successfully identified from a very small number of training samples. However, as the set of target languages was fixed and the training samples were selected manually, we searched for another benchmark to obtain more relevant results. This is often achieved by using randomly generated training and testing samples.

   There was an approach [35] for evaluating ABR inference algorithms able to supply random training and testing samples based on a given grammar. Unfortunately, we showed that the method has several drawbacks, and we also proved some theoretical limits of similar grammar-based benchmarks.

   As we needed some more significant test, we proposed a new benchmark suitable for ABR inference methods. It is based on randomly generated RRW-automata which we used to generate random training and testing samples.

   This new benchmark was then used to evaluate our inference algorithm. The results of those tests confirmed the expected dependency of the quality of the inferred models on both the number of training samples and the considered reversibility level of the models. The overall results of our method

were very impressive, and in the case of high number of training samples and also high reversibility level, they were nearly excellent.

It is very interesting to see that those results were obtained despite the fact that the search space contained only S-kR-RRW-automata (i.e. our proposed model) that are quite restricted when compared to RRW-automata used to represent the target languages in benchmarks.

The main outcomes of this thesis can be summarized as it follows:

1. the theorems stating the complexity of the problem of inferring the ABR,
2. the proposed model (S-kR-RRWW-automata) and the analysis of its properties,
3. the inference algorithm for learning the ABR,
4. the negative results on benchmarks based on grammars,
5. the benchmark for ABR inference algorithms based on restarting automata, and
6. the evaluation of the proposed algorithm for learning the ABR.

From the notes presented above, it is obvious that the goals set in Chapter 3 were achieved. There are several ways to extend the results of this thesis:

- We infer only *single* restarting automata. This way, the inference task is simplified as we do not have to decide whether two reductions performing the same rewrite should be handled by different meta-instructions or by the same meta-instruction. Though the results achieved with single restarting automata were really good, it would be helpful to allow multiple meta-instructions sharing the same rewrite. To partition the reduction samples among them, one can use e.g. the idea of Occam's razor in the case of learning from positive samples only. If there are negative samples available, they could be used to check whether particular reductions can be handled by the same rewriting meta-instruction (a similar technique was used in [58, 30]).
- It would be interesting to see the performance of our approach on samples from some natural language. Note that it does not suffice to supply just sample sentences but whole FPABR samples are needed. It could be possible to start with some subset of that natural language if there are not enough samples available. Prague Dependency Treebank could be possibly employed [62].
- We could include negative samples into the input. They can be used to tune the parameters of the considered models, particularly to decide when increasing their power is needed.

Overall, this thesis contributes greatly to the area of learning the ABR from samples. It deals with the whole range of related topics. It formalizes the definition of the ABR inference problem, analyzes its complexity, and proposes and evaluates an ABR inference algorithm. According to the benchmarks, the proposed method can be successfully used for machine learning of the ABR.

Bibliography

[1] AHO, A. V.; HOPCROFT, J. E.; ULLMAN, J. D. *The design and analysis of computer algorithms*. Reading, Massachusetts : Addison-Wesley, 1974, 470 p. ISBN 0201000296.

[2] AMAR, V.; PUTZOLU, G. On a family of linear grammars. *Information and Control*, 1964, vol. 7, issue 3, pp. 283–291.

[3] ANGLUIN, D. Inductive inference of formal languages from positive data. *Information and Control*, 1980, vol. 45, issue 2, pp. 117–135.

[4] ANGLUIN, D. Inference of reversible languages. *Journal of ACM*, 1982, vol. 29, no. 3, pp. 741–765.

[5] ANGLUIN, D. Learning regular sets from queries and counterexamples. *Information and Computation*, 1987, vol. 75, issue 2, pp. 87–106.

[6] ANGLUIN, D. Negative results for equivalence queries. *Machine Learning*, 1990, vol. 5, issue 2, pp. 121–150.

[7] ANGLUIN, D. On the complexity of minimum inference of regular sets. *Information and Control*, 1978, vol. 39, issue 3, pp. 337–350.

[8] ANGLUIN, D. Queries and concept learning. *Machine Learning*, 1988, vol. 2, issue 4, pp. 319–342.

[9] BAKER, B. S.; BOOK, R. V. Reversal-bounded multi-pushdown machines. In *13th Annual Symposium on Switching and Automata Theory (SWAT 1972)*. IEEE, 1972. pp. 207–211. ISSN 0272-4847.

[10] BASOVNÍK, S; MRÁZ, F. Learning limited context restarting automata by genetic algorithms. In DASSOW, J.; TRUTHE, B. (eds.). *Proceedings of 21. Theorietag Automaten und Formale Sprachen (Allrode im Harz, Germany)*. Magdeburg : Otto-von-Guericke-Universität, 2011. pp. 1-4.

[11] CHOMSKY, N. Three models for the description of language. *IRE Transactions on Information Theory*, 1956, vol. 2, issue 3, pp. 3113–124.

[12] ČEJKA, J. *Learning correctness preserving reduction analysis*. Bc. project, Faculty of Mathematics and Physics, Charles University, 2003.

[13] ČEJKA, J. *Učení redukční analýzy*. Master's thesis, Faculty of Mathematics and Physics, Charles University, 2007. 111 p.

[14] ČERNO, P. Clearing restarting automata and grammatical inference. In HEINZ, J.; HIGUERA, C. de la; OATES, T. (eds.). *JMLR Workshop and Conference Proceedings, Volume 21: ICGI 2012 : Proceedings of the Eleventh International Conference on Grammatical Inference September 5-8, 2012, University of Maryland, College Park, United States*. 2012. pp. 54–68.

[15] ČERNO, P.; MRÁZ, F. Δ-clearing restarting automata and CFL. In MAURI, G.; LEPORATI, A. *Developments in Language Theory: 15th International Conference, DLT 2011, Milan, Italy, July 19-22, 2011 : proceedings*. Berlin : Springer, 2011. pp. 153-164. (Lecture notes in computer science; 6795). ISBN 9783642223211.

[16] ČERNO, P.; MRÁZ, F. Clearing restarting automata. *Fundamenta Informaticae*, 2010, vol. 104, no. 1-2. pp. 17–54.

[17] DAHLHAUS, E.; WARMUTH, M. K. Membership for growing context-sensitive grammars is polynomial. *Journal of Computer System Sciences*, 1986, vol. 33, issue 3, pp. 456–472.

[18] DUPONT, P. Regular grammatical inference from positive and negative samples by genetic search: the GIG method. In CARRASCO, R. C.; ONCINA, J. (eds.). *Grammatical Inference and Applications: Second International Colloquium, ICGI-94, Alicante, Spain, September 21-23, 1994: proceedings*. Berlin : Springer, 1994. pp. 236–245. (Lecture notes in artificial intelligence; 862). ISBN 3540584730.

[19] DUPONT, P. *Utilisation et apprentissage de modèles de language pour la reconnaissance de la parole continue.* PhD thesis. Paris : École Normale Supérieure des Télécommunications, 1996.

[20] DUPONT, P.; MICLET, L.; VIDAL, E. What is the search space of the regular inference? In CARRASCO, R. C.; ONCINA, J. (eds.). *Grammatical Inference and Applications: Second International Colloquium, ICGI-94, Alicante, Spain, September 21-23, 1994: proceedings*. Berlin : Springer, 1994. pp. 25–37. (Lecture notes in artificial intelligence; 862). ISBN 3540584730.

[21] EYRAUD, R.; HIGUERA, C. de la; JANODET, J.-C. LARS: A learning algorithm for rewriting systems. *Machine Learning*, 2007, vol. 66, no. 1, pp. 7–31.

[22] GARCÍA, P.; VIDAL, E. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1990, vol. 12, no. 9, pp. 920–925.

[23] GOLD, E. M. Complexity of automaton identification from given data. *Information and Control*, 1978, vol. 37, issue 3, pp. 302–320.

[24] GOLD, E. M. Language identification in the limit. *Information and Control*, 1967, vol. 10, issue 5, pp. 447–474.

[25] HARTWIG, M. On the density of regular and context-free languages. In THAI, M. T.; SAHNI, S. *Computing and Combinatorics : 16th Annual International Conference, COCOON 2010, Nha Trang, Vietnam, July 19-21, 2010 : proceedings*. Berlin : Springer, 2010. pp. 318–327. (Lecture notes in computer science; 6196). ISBN 9783642140303.

[26] HIGUERA, C. de la. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 1997, vol. 27, no. 2, pp. 125–138.

[27] HIGUERA, C. de la. Grammatical inference : learning automata and grammars. Cambridge : Cambridge university press, 2010. 417 p. ISBN 9780521763165.

[28] HINGSTON, P. A genetic algorithm for regular inference. In SPECTOR, L. ... [et al.] (eds.). *GECCO-2001: proceedings of the Genetic and Evolutionary Computation Conference : a joint meeting of the sixth annual Genetic Programming Conference (GP-2001) and the tenth International Conference on Genetic Algorithms (ICGA-2001) : July 7-11, 2001, San Francisco, California*. San Francisco : Morgan Kaufmann, 2001. pp. 1299–1306. ISBN 1558607749.

[29] HOFFMANN, P. Evaluation of analysis by reduction inference algorithms. In *AIA 2010 – Artificial Intelligence and Applications*. Calgary : Acta Press, 2010. pp. 67–74. ISBN 9780889868175.

[30] HOFFMANN, P. Improving RPNI algorithm using minimal message length. In *Proceedings of the 25th IASTED International Multi-Conference: Artificial Intelligence and Applications, AIAP'07*. Anaheim : Acta Press, 2007. pp. 378–383. ISBN 9780889866294.

[31] HOFFMANN, P. Learning analysis by reduction from positive data using reversible languages. In *Proceedings of the 2008 Seventh International Conference on Machine Learning and Applications, ICMLA '08*. Washington : IEEE Computer Society, 2008. pp. 141–146. ISBN 9780769534954.

[32] HOFFMANN, P. Learning restarting automata by genetic algorithms. In BIELIKOVÁ, M. (ed.). *SOFSEM 2002: Student research forum, Milovy, Czech Republic*. Milovy : Slovak technical university, 2002. pp. 15–20.

[33] HOFFMANN, P. *Restarting automata inference complexity*. Technical Report TR 2007/9. Prague : Faculty of Mathematics and Physics, Charles University, 2007. 10 p.

[34] HOFFMANN, P. *Učenie reštartovacích automatov genetickými algoritmami*. Bachelor's thesis. Prague : Faculty of Mathematics and Physics, Charles University, 2000. [*In Czech, the English translation of the title is Learning restarting automata by genetic algorithms.*]

[35] HOFFMANN, P. *Učenie reštartovacích automatov genetickými algoritmami*. Master's thesis. Prague : Faculty of Mathematics and Physics, Charles University, 2003. 129 p. [*In Czech, the English translation of the title is Learning restarting automata by genetic algorithms.*]

[36] HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. *Introduction to automata theory, languages, and computation*. 2nd ed. Reading : Addison-Wesley, 2001. 521 p. ISBN 0201441241.

[37] JANČAR, P.; MRÁZ, F.; PLÁTEK, M.; VOGEL, J. On monotonic automata with a restart operation. *Journal of Automata, Languages and Combinatorics*, 1999, vol. 4, no. 4, pp. 287–311.

[38] JANČAR, P.; MRÁZ, F.; PLÁTEK, M.; VOGEL, J. On restarting automata with rewriting. In PĂUN, G.; SALOMAA, A. *New Trends in Formal Languages : Control, Cooperation, and Combinatorics*. Berlin : Springer, 1997. pp. 119–136. (Lecture notes in computer science; 1218). ISBN 9783540628446.

[39] JANČAR, P.; MRÁZ, F.; PLÁTEK, M.; VOGEL, J. Restarting automata. In REICHEL, H. (ed.). *Fundamentals of Computation Theory, 10th International Symposium, FCT '95, Dresden, Germany, August 22-25, 1995 : proceedings*. Berlin : Springer, 1995. pp. 283–292. (Lecture Notes in Computer Science; 965). ISBN 3540602496.

[40] KARP, R. Reducibility among combinatorial problems. In MILLER, R.; THATCHER, J. (eds.). *Complexity of Computer Computations*. New York : Plenum Press, 1972. pp. 85–103. ISBN 0306307073.

[41] KEARNS, M. J.; VALIANT, L. G. Cryptographic limitations on learning boolean formulae and finite automata. In DAVID, S. J. (ed.). *Proceedings of the twenty-first annual ACM symposium on Theory of computing, STOC'89, Seattle, Washington, USA*. New York : ACM, 1989. pp. 433–444. ISBN 0897913078.

[42] KOBAYASHI, S.; YOKOMORI, T. Learning concatenations of locally testable languages from positive data. In ARIKAWA, S. A.; JANTKE, K. P. *Algorithmic Learning Theory, 4th International Workshop on Analogical and Inductive Inference, AII '94, 5th International Workshop on Algorithmic Learning Theory, ALT '94, Reinhardsbrunn Castle, Germany, October 10-15, 1994: proceedings*. Berlin : Springer, 1994. pp. 407–422. (Lecture Notes in Computer Science; 872). ISBN 3540585206.

[43] LANG, K. J. Random DFA's can be approximately learned from sparse uniform examples. In *Proceedings of the fifth annual workshop on Computational learning theory, COLT '92, Pittsburgh, Pennsylvania, USA*. New York : ACM, 1992. pp. 45–52. ISBN 089791497X.

[44] LANG, K. J.; PEARLMUTTER, B. A. *The Abbadingo One DFA learning competition* [online]. c2007. [cit. 2011-05-21]. Available from www: <http://www-bcl.cs.may.ie>.

[45] LANG, K. J.; PEARLMUTTER, B. A.; PRICE, R. A. Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In HONAVAR, V.; SLUTZKI, G. (eds.). *Grammatical inference : 4th International Colloquium, ICGI-98, Ames, Iowa, USA, July 12-14, 1998: proceedings*. Berlin : Springer, 1998. pp. 1– 12. (Lecture notes in artificial intelligence; 1433). ISBN 9783540647768 .

[46] LANGHOLM, T.; BEZEM, M. A descriptive characterisation of even linear languages. *Grammars*, 2003, vol. 6, issue 3, pp. 169–181.

[47] LEVENSTEIN, V. I. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 1966, vol. 10, no. 8, pp. 707–710.

[48] LINZ, P. *An introduction to formal languages and automata*. Lexington : D. C. Heath, 1990. 373 p. ISBN 0669173428.

[49] LOPATKOVÁ, M.; PLÁTEK, M; KUBOŇ, V. Modelling syntax of free word-order languages: dependency analysis by reduction. In MATOUŠEK, V.; MAUTNER, P.; PAVELKA, T. (eds.). *Text, speech and dialogue: 8th international conference, TSD 2005, Karlovy Vary, Czech Republic, September 12 – 15, 2005: proceedings*. Berlin : Springer, 2005. pp. 140–147. (Lecture notes in computer science; 3658). ISBN 9783540287896.

[50] MAURER, H. A; SALOMAA, A.; WOOD, D. On generators and generative capacity of EOL forms. *Acta Informatica*, 1980, vol. 13, issue 1, pp. 87–107.

[51] McNAUGHTON, R. The loop complexity of pure-group events. *Information and Control*, 1967, vol. 11, issue 1–2, pp. 167–176.

[52] MESSERSCHMIDT, H.; MRÁZ, F.; OTTO, F.; PLÁTEK, M. *On the descriptional complexity of simple RL-automata*. Technical Report 4/06. Kässel : Universität Kässel, 2006. 13 pages.

[53] MICLET, L. *Inférence de grammaires régulières*. PhD thesis. Paris : E.N.S.T., 1979.

[54] MITCHELL, T. M. *Machine learning*. Boston : McGraw-Hill, 1997. 414 p. ISBN 0070428074.

[55] MRÁZ, F.; OTTO, F.; PLÁTEK, M. Learning analysis by reduction from positive data. In SAKAKIBARA, Y. ... [et al.] (eds.). *Grammatical inference: algorithms and applications: 8th international conference, ICGI 2006, Tokyo, Japan, September 20 – 22, 2006: proceedings*. Berlin : Springer, 2006. pp. 125–136. (Lecture notes in computer science; 4201). ISBN 3540452648.

[56] MRÁZ, F.; PLÁTEK, M.; PROCHÁZKA, M. On special forms of restarting automata. *Grammars*, 1999, vol. 2, issue 3, pp. 223–233.

[57] NATARJAN, B. K. *Machine learning: a theoretical approach*. San Mateo : Morgan Kaufmann, 1991. 217 p. ISBN 1558601481.

[58] ONCINA, P.; GARCÍA, J. Inferring regular languages in polynomial update time. In SANFELIU, A.; PÉREZ de la BLANCA, N.; VIDAL, E. *Pattern recognition and image analysis : selected papers from the IVth Spanish Symposium*. Singapore : World Scientific, 1992. pp. 49–61. ISBN 9810208812.

[59] OTTO, F. Restarting automata. In ÉSIK, Z.; MARTIN-VIDE, C.; MITRANA, V. *Recent Advances in Formal Languages and Applications*. Berlin : Springer, 2006. pp. 269–303. ISBN 9783540334606.

[60] OTTO, F. Restarting automata and their relation to the Chomsky hierarchy. In ÉSIK, Z.; FÜLÖP, Z. (eds.). *Developments in language theory: 7th international conference, DLT 2003, Szeged, Hungary, July 7 – 11, 2003: proceedings*. Berlin : Springer, 2003. pp. 55–74. (Lecture notes in computer science; 2710). ISBN 3540404341.

[61] PAREKH, R.; HONAVAR, V. Learning DFA from simple examples. In LI, M.; MARUOKA, A. (eds.). *Algorithmic Learning Theory: 8th International Workshop, ALT '97, Sendai, Japan, October 6-8, 1997: proceedings*. Berlin : Springer, 1997. pp. 116–131. (Lecture notes in computer science; 1316). ISBN 9783540635772.

[62] *PDT: Prague dependency treebank 2.5.* Praha : Univerzita Karlova, Ústav formální a aplikované lingvistiky, 2011. Available from www: <http://ufal.mff.cuni.cz/pdt2.5/>.

[63] PITT, L.; WARMUTH, M. K. The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of ACM*, 1993, vol. 40, no. 1, pp. 95–142.

[64] PLÁTEK, M.; LOPATKOVÁ, M.; OLIVA, K. Restarting automata: motivations and applications. In HOLTZER, M. (ed.). *Workshop 'Petrinetze' and 13. Theorietag 'Formale Sprachen und Automaten': proceedings*. München : Institut für Informatik, Technische Universität, 2003. pp. 90–96.

[65] RICH, E. A. *Automata, computability and complexity: theory and applications*. Pearson Prentice Hall, 2007. 1120 p. ISBN 9780132288064.

[66] SEMPERE, J. M.; GARCÍA, P. A characterization of even linear languages and its application to the learning problem. In CARRASCO, R. C.; ONCINA, J. *Grammatical Inference and Applications: Second International Colloquium, ICGI-94, Alicante, Spain, September 21-23, 1994: proceedings*. Berlin : Springer, 1994. pp. 38–44. (Lecture notes in computers science; 862). ISBN 9783540584735.

[67] SEMPERE, J. M.; GARCÍA, P. Learning locally testable even linear languages from positive data. In ADRIAANS, P.; FERNAU, H.; ZAANEN, M. (eds.). *Grammatical Inference: Algorithms and Applications : 6th International Colloquium, ICGI 2002, Amsterdam, The Netherlands, September 23-25, 2002: proceedings*. Berlin : Springer, 2002. pp. 225–236. (Lecture notes in computer science; 2484). ISBN 3540442391.

[68] TOMITA, M. Dynamic construction of finite automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Cognitive Science Conference*. Ann Arbor, 1982. pp. 105–108.

[69] TRAKHTENBROT, B. A.; BARZDIN, IA. M. *Finite automata : behavior and synthesis*. New York : American Elsevier, 1973. 321 p. ISBN 0444104186.

[70] VALIANT, L. G. A theory of the learnable. *Communications of the ACM*, 1984, vol. 27, no. 11, pp. 1134–1142.

[71] WAGNER, K.; WECHSUNG, G. *Computational complexity*. Dordrecht : D. Reidel Publishing Company, 1985. 551 p. ISBN 9027721467.

[72] YOKOMORI, T. *A note on the polynomial-time identification of strictly local languages in the limit*. Technical Report CSIM90-03. Tokyo : Department of Computer Science and Information Mathematics, University of Electro-Communications, 1990.

[73] YOKOMORI, T.; KOBAYASHI, S. Learning local languages and their application to DNA sequence analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998, vol. 20, no. 10, pp. 1067–1079.

[74] YU, S. Regular languages. In SALOMAA, A.; ROZENBERG, G. *Handbook of Formal Languages. Volume 1: Word language grammar*. Berlin : Springer, 1997. pp. 41–110. ISBN 3540604200.

List of Symbols

Sets and operations on sets	
$\mathbb{N}$	the set of all natural numbers (including zero)
$\subseteq$	the subset relation
$\subset$	the proper subset relation
$\subset_{\mathrm{fin}}$	the finite subset relation
$X \cup Y$	the union of the sets X and Y
$X \cap Y$	the intersection of the sets X and Y
$X \setminus Y$	the difference of the sets X and Y
$X \oplus Y$	the symmetric difference of the sets X and Y
$\mathrm{P}(X)$	the set of all subsets of the set X

Words and operations on words	
λ	the empty word
$\mathrm{b}(n)$	the word 0^n
$\mathrm{B}(X)$	the set of values $\mathrm{b}(n)$ for all $n \in X$
$\lvert w \rvert$	the length of the word w
$\lvert w \rvert_a$	the number of occurrences of the symbol a in the word w
$w_{\overline{i_0,\ldots,i_n}}$	the word obtained from w by removing the symbols at specified positions (starting from zero)
w^{R}	the reverse of the word w
$u \cdot v$	the concatenation of the words u and v (denoted also uv)
uv	the concatenation of the words u and v (denoted also $u \cdot v$)
$w \lll n$	the rotation of the word w by n symbols to the left
$\mathrm{rot}(w)$	the set of all rotations of the word w to the left
w^{f}	the folding of the word w

Operations on languages	
$L \cdot L'$	the concatenation of the languages L and L'
L^n	the language obtained by n-times concatenating the language L with itself
L^*	Kleene closure of the language L
L^+	the union of L^n for all $n > 0$
$\Sigma^{\geq k}$	the set of all words over the alphabet Σ of length at least k
L^{c}	the complement of the language L
L^{R}	the reverse of the language L
$\mathrm{Pr}(L)$	the set of all prefixes of all words from the language L
$\mathrm{LQ}(L, u)$	the left quotient of the language L and the word u
$\mathrm{LQ}(L, L')$	the left quotient of the language L with the language L'
$\mathrm{RQ}(L, L')$	the right quotient of the language L with the language L'
$\mathrm{PTA}(S)$	the prefix tree acceptor for the language S
Σ_{f}	the folding of the alphabet Σ
L^{f}	the folding of the language L

Symbols related to automata and grammars in general	
$\mathrm{L}(A)$	the language accepted by the automaton A
$\mathrm{L}(G)$	the language generated by the grammar G
$\mathrm{L}(e)$	the language represented by the regular expression e
$\mathcal{A}(X)$	the class of all automata of the type X
$\mathcal{L}(X)$	the class of all languages accepted by automata of the type X
G_{f}	the folding of the even linear grammar G

Symbols related to finite state automata	
$\mathrm{A}(L)$	the canonical acceptor for the regular language L
$\mathrm{L}(e)$	the language represented by the regular expression e
A/π	the quotient of the finite state automaton A and the partition of its states π
A^{R}	the reverse of the finite state automaton A
δ^{R}	the reverse of the transition function δ

Symbols related to restarting automata	
$\vdash_M$	the reduction relation of the restarting automaton M
$\vdash_M^*$	the reflexive and transitive closure of the reduction relation of the restarting automaton M
$\mathrm{L_C}(M)$	the characteristic language accepted by the restarting automaton M
$\dot{\mathrm{R}}(\Gamma)$	the set of all possible located reductions over the alphabet Γ
$\overline{\mathrm{R}}(\Gamma)$	the set of all possible sliding reductions over the alphabet Γ
L_k^f	the set of languages accepted by all RRW-automata having size (according to the measure f) at most k

List of Tables

List of Abbreviations

ABR	analysis by reduction
AF-measure	the type of a function used to measure given RRW-automata
CFL	the class of all context-free languages
CSL	the class of all context-sensitive languages
DFSA	deterministic finite state automaton
EL	the class of all even linear languages
FPABR sample	fully positive analysis by reduction sample
FSA	finite state automaton
GCSL	the class of all growing context-sensitive languages
k-RI	the learning algorithm for k-reversible languages proposed in [4]
k-SLT-R	the class of all strictly k-testable restarting automata
kR-RRWW-automaton	k-reversible restarting automaton
LARS	the learning algorithm for rewriting systems proposed in [21]
LIN	the class of all linear languages
MAT	minimal adequate teacher
NFSA	non-deterministic finite state automaton
NP	the class of all decision problems solvable by a non-deterministic Turing machine in polynomial time
PAC model	probably approximately correct model
PTA	prefix tree acceptor
RPNI	the learning algorithm called *Regular Positive and Negative Inference*, proposed in [58]
RR-automaton	restarting automaton without rewriting
RRW-automaton	restarting automaton without auxiliary symbols
RRWW-automaton	restarting automaton
S-kR-RRWW-automaton	single k-reversible restarting automaton (similarly for types RRW and RR)
S-RRWW-automaton	*single* restarting automaton
S-ZR-RRWW-automaton	*single* zero-reversible restarting automaton
S-*R-RRWW	the union of all classes of S-kR-RRWW-automata (analogically for RRW-automata and RR-automata)
SLT-R	the class of all strictly locally testable restarting automata
ZR	the learning algorithm for 0-reversible languages proposed in [4]
0-RI	the learning algorithm for 0-reversible languages proposed in [4] (the same as ZR)

Attachments

The enclosed CD contains the following files:

- notes.pdf contains this thesis.
- experiments.ods contains a summary of results achieved by the algorithm Omega* in benchmarks described in Subsection 7.3.2.