Free Word-Order and Restarting Automata

FRANTIŠEK MRÁZ
Charles University, Faculty of Mathematics and Physics
Department of Computer Science, Malostranské nám. 25
118 00 Praha 1, Czech Republic
E-mail: mraz@ksvi.ms.mff.cuni.cz

FRIEDRICH OTTO
Fachbereich Mathematik/Informatik
Universität Kassel, 34109 Kassel, Germany
E-mail: otto@theory.informatik.uni-kassel.de

MARTIN PLÁTEK
Charles University, Faculty of Mathematics and Physics
Department of Computer Science, Malostranské nám. 25
118 00 Praha 1, Czech Republic
E-mail: Martin.Platek@mff.cuni.cz

Abstract
In natural languages with a high degree of word-order freedom syntactic phenomena like dependencies (subordinations) or valences do not depend on the word-order (or on the individual positions of the individual words). This means that some permutations of sentences of these languages are in some (important) sense syntactically equivalent. Here we study this phenomenon in a formal way. Various types of \( j \)-monotonicity for restarting automata can serve as parameters for the degree of word-order freedom and for the complexity of word-order in sentences (languages). Here we combine two types of parameters on computations of restarting automata:

1. the degree of \( j \)-monotonicity, and
2. the number of rewrites per cycle.

We study these notions formally in order to obtain an adequate tool for modelling and comparing formal descriptions of (natural) languages with different degrees of word-order freedom and word-order complexity.

1 Introduction
The original motivation for introducing the restarting automaton was the desire to model the so-called \textit{analysis by reduction} of natural languages. Many aspects of the work on restarting automata are motivated by the basic tasks of computational linguistics (e.g., devising multilevel language descriptions) as well as by applied tasks

---

\(^1\)The work of the first two authors was supported by a grant from the Deutsche Forschungsgemeinschaft. The third author was partially supported by the Grant Agency of the Czech Republic under Grant-No. 201/04/2102 and by the program ‘Information Society’ under project 1ET100300517. This work was performed while F. Mráz was visiting the University of Kassel.
(e.g., constructing grammar checkers for free word-order languages [14]). More about the motivation and about the corresponding literature can be found in [15, 17].

From a theoretical point of view the restarting automaton can be seen as a tool that yields a very flexible generalization of analytical grammars, that is, in a very flexible way it introduces a basic syntactic system (an approximation to the formalization of the analysis by reduction), which contains the full information about the input vocabulary (set of wordforms), the categorial vocabulary, the set of reductions (rewritings), the recognized language, the language of sentential forms, and the categorial language. On the other hand, the restarting automaton can be considered as a generalization and a refinement of the pushdown automaton (see, e.g., [16]).

Various restricted versions of restarting automata and various constraints for them are considered in the literature. In particular, a monotonicity constraint has been introduced for restarting automata which is based on the idea that from one rewrite operation to the next within a computation, the actual place where the rewriting is performed must not increase its distance from the right end of the tape. Monotone restarting automata essentially model bottom-up context-free analyzers. Accordingly, it has been shown that monotone restarting automata (with auxiliary symbols) characterize the class \( \text{CFL} \) of context-free languages, and various restricted versions of deterministic monotone restarting automata (with or without auxiliary symbols) characterize the class \( \text{DCFL} \) of deterministic context-free languages [9].

Also a generalization of the constraint of monotonicity has been considered, which models the generalization from bottom-up one-pass parsers to bottom-up multi-pass parsers. For an integer \( j \geq 2 \), a computation is called \( j \)-monotone if the corresponding sequence of rewrite steps can be partitioned into at most \( j \) interleaved subsequences such that each of these subsequences is monotone. It was shown that by increasing the value of the parameter \( j \), the expressive power of restarting automata without auxiliary symbols is increased [11].

Here we introduce and use a new type of restarting automaton, the freely rewriting restarting automaton, \( \text{FRR-automaton} \), which may perform an unlimited number of rewrite operations per cycle, in order to classify the word-order of natural languages. In fact, we propose an infinite hierarchy of classes of basic syntactic systems and of classes of languages of sentential forms (of restarting automata) from the points of view of word-order complexity and word-order freedom. Our main goal is to show that the word-order freedom (together with some type of valences (dependencies)) can cause complex syntactic phenomena. Similar considerations were made before using dependency grammars [8], however our approach is more general, and it can be applied to various types of grammars, including Chomsky, categorial, pure, Marcus, and dependency grammars, using simulations by restarting automata. The word-order constraints play an important role in modern computational linguistics (see, e.g., [5]). We only consider restarting automata, which are not stronger (with respect to their weak generative capacity) than linear-bounded automata. In order to obtain the intended results certain combinations of constraints for \( \text{FRR-automata} \) are studied.

The complexity of the word-order of natural languages can be illustrated by several constructions found in some languages. The first two samples show non-context-free
constructions in languages which are considered to be languages with fixed word-order. The next three samples are taken from languages with a considerable degree of word-order freedom.

In [3] Bresnan et al. give the following example from Dutch:

\( \text{(dat) Jan Piet Marie de kinderen zag helpen laten zwemmen.} \)

\( \text{[(that)-Jan-Piet-Marie-the-children-saw-help-make-swim.]} \)

\( \text{[(that) Jan saw Piet help Marie make the children swim.]} \)

This example shows a duplication-like structure of the form \( w\overline{w} \), where \( \overline{w} \) is the word obtained from \( w \) by replacing each symbol by its barred copy. Using analysis by reduction we would like to get the following sequence of reductions:

\( \text{(dat) Jan Piet Marie de kinderen zag helpen laten zwemmen.} \)

\( \text{(dat) Jan Piet Marie zag helpen VG\text{inf}.} \)

\( \text{(dat) Jan Piet zag VG\text{dat}.} \)

Here the rewritten parts are in bold font. In the first step, the noun phrase ‘de kinderen’ (\textit{the children}) and the infinitival verb complement ‘laten zwemmen’ are replaced by the category VG\text{inf}, which means that an infinitive construction was deleted. Note that the rewritten part is not contiguous. In the second step, the words ‘Marie’, ‘helpen,’ and the category VG\text{inf} are replaced by a category VG\text{dat}, which represents a subordinate clause.

Similar constructions, where an adequate analysis by reduction requires rewriting words which are in distant parts of a sentence, can be found in many other languages. Shieber found the following construction in the Zürich dialect of German [19]:

Jan säit das mer d’chind em Hans es huus haend wele laa hälfte aastrüche.

\( \text{[Jan said that we the children Hans the house wanted to let help paint.]} \)

\( \text{[Jan said that we wanted to let the children help Hans paint the house.]} \)

It has the structure \( xwa^mb^nyc^md^nz \), where \( a, b \) stand for dative and accusative noun phrases, respectively, and \( c, d \) for the corresponding dative and accusative verb phrases, respectively.

Analysis by reduction for the above two sample sentences (more precisely their generalized forms) can be modelled by 2-monotone FRR-automata with two rewrites per cycle which we call 2-constrained. The degree of \( j \)-constrainability will serve as a synonym for word-order complexity. In this way we enrich the taxonomy of various word-order constraints given in [8].

In German the following are correct (parts of) sentences [18]:

\( \cdot \) ... daß Peter dem Kunden den Kühlschrank zu reparieren zu helfen versucht.

\( \cdot \) ... that Peter the client the refrigerator to repair to help tries.

\( \cdot \) ... that Peter tries to help the client to repair the refrigerator.

\( \cdot \) ... daß Peter versucht, dem Kunden zu helfen, den Kühlschrank zu reparieren.
• ...daß Peter versucht, dem Kunden den Kühlschrank zu reparieren zu helfen.

Even more correct sentences can be obtained by permuting five elements (delimited by < and >) in the following example taken from [7]:

... daß <eine hiesige Firma> <meinem Onkel> <die Möbel> <vor drei Tagen> <ohne Voranmeldung> zugestellt hat.

[... that-a-local-company-delivered-the-furniture-to-my-uncle-three-days-ago-without-advance-notice-delivered-has.]

For a more detailed discussion on word-order of German see [18].

In some other languages like most Slavonic languages, the syntactic relations between words are specified by other means than the position in a sentence – by a rich inflexion. Often all permutations of the words in a clause are possible. All the following sentences are correct in Czech:

• Všichni prošli kursem.
  [All-passed-the-course.]
• Kursem všichni prošli.
• Kursem prošli všichni.
• Prošli všichni kursem.
• Prošli kursem všichni.
• Všichni kursem prošli.

In the last three examples some words in each sentence can be (almost) freely permuted. This resembles the transformational grammar approach [4], where transformations are used to increase the descriptive power of context-free grammars. We will deal with permutations only and say that all sentences which differ in the word-order only are Parikh equivalent (that is, they contain the same number of occurrences of each word). Based on the Parikh equivalence we introduce a reduction equivalence of sentences and study the degree of word-order freedom of languages.

The paper is structured as follows. In the next section we present definitions of FRR-automata and their \( j \)-constrained computations. The language \( L_{SF}(M) \) accepted by an FRR-automaton \( M \) is called a language of sentential forms. If we restrict the set of possible input symbols to a fixed subset \( \Sigma \) of the working alphabet, we say that the automaton \( M \) accepts the input language \( L_I = L_{SF} \cap \Sigma^* \). In Section 3 we demonstrate the power of \( j \)-constrained computations of FRR-automata: the languages of sentential forms accepted by \( j \)-constrained computations of FRR-automata create an infinite hierarchy. Further, for each degree \( j \) of constrainability,
the class of languages of sentential forms accepted by \( j \)-constrained computations of FRR-automata is a proper subclass of the class of input languages accepted by \( j \)-constrained computations of FRR-automata. In other words: the use of auxiliary (non-input) symbols can hide the real complexity of the word-order. Then we formalize the degree of word-order freedom of a language (or an FRR-automaton) through the notion of \( j \)-scalability of FRR-automata. Roughly speaking, an FRR-automaton is \( j \)-scalable if increasing constrainability of its computations from \( i \) to \( i + 1 \), for all \( 1 \leq i < j \), extends the set of accepted words only by words obtained by permutations of symbols in words accepted already by \( i \)-constrained computations, and simultaneously the structure of reductions defined by cycles of \( M \) remains the same (up to the Parikh equivalence). Finally, in Section 4 we shortly discuss the achieved results and state some problems for further research on this subject.

2 Definitions

Throughout the paper we will use \( \lambda \) to denote the empty word. Further, \(|w|\) will denote the length of the word \( w \), and if \( a \) is an element of the underlying alphabet, then \(|w|_a\) denotes the \( a \)-length of \( w \), that is, the number of occurrences of the letter \( a \) in \( w \). Further, \( \mathbb{N}_+ \) will denote the set of all positive integers.

We start by describing the model of the restarting automaton we are going to use in this paper.

A freely rewriting restarting automaton, FRR-automaton for short, is a (nondeterministic) machine that is described by a 7-tuple \( M = (Q, \Gamma, \mathcal{Q}, \mathcal{C}, \mathcal{L}, \mathcal{R}, \delta) \), where \( Q \) is a finite set of states, \( \Gamma \) is a finite tape alphabet, \( \mathcal{Q}, \mathcal{C}, \mathcal{L} \in \Gamma \) are symbols that are used as markers for the left and right border of the work space, respectively, \( q_0 \in Q \) is the initial state, \( k \geq 1 \) is the size of the read/write window, and \( \delta \) is the transition relation that associates to each pair \((q, u)\) consisting of a state \( q \) and a possible content \( u \) of the read/write window a finite set of possible transition steps. There are four types of transition steps:

1. A move-right step (MVR) causes \( M \) to shift the read/write window one position to the right and to change the state. However, the read/write window cannot move across the right sentinel \( \mathcal{L} \).

2. A rewrite step causes \( M \) to replace a non-empty prefix \( u \) of the content of the read/write window by a string \( v \), thereby possibly changing the length of the tape, and to change the state. Further, the read/write window is placed immediately to the right of the string \( v \). However, occurrences of the delimiters \( \mathcal{Q} \) and \( \mathcal{L} \) can neither be deleted nor newly created by a rewrite step.

3. A restart step causes \( M \) to place its read/write window over the left end of the tape, so that the first symbol it sees is the left sentinel \( \mathcal{Q} \), and to reenter the initial state \( q_0 \).

4. An accept step causes \( M \) to halt and accept.
If $\delta(q,u) = \emptyset$ for some pair $(q,u)$, then $M$ necessarily halts, and we say that $M$ rejects in this situation. In addition, it is required that there exists a weight function $\omega : \Gamma \to \mathbb{N}_+$ such that, for each rewrite operation $u \rightarrow v$, $\omega(v) < \omega(u)$ holds. Here $\omega$ is extended to a morphism $\omega : \Gamma^* \to \mathbb{N}$ by taking $\omega(\lambda) := 0$ and $\omega(wa) := \omega(w) + \omega(a)$ for all $w \in \Gamma^*$ and all $a \in \Gamma$. Thus, the FRR-automaton is a variant of the shrinking restarting automaton considered in [13].

A configuration of $M$ is a string $q\alpha\beta$, where $q \in Q$, and either $\alpha = \lambda$ and $\beta \in \{ \|$ or $\alpha \in \{ q \} \cdot \Gamma^* \cdot \{ \|$ or $\alpha \in \{ q \} \cdot \Gamma^* \cdot \{ \} \}$; here $q$ represents the current state, $\alpha\beta$ is the current content of the tape, and it is understood that the window contains the first $k$ symbols of $\beta$ or all of $\beta$ when $|\beta| \leq k$. A restarting configuration is of the form $q_0\alpha\|$.

We observe that any computation of $M$ consists of certain phases. A phase, called a cycle, starts in a restarting configuration, the head moves along the tape performing move-right and rewrite operations until a restart operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, the computation necessarily finishes in a halting configuration – such a phase is called a tail. It is required that in each cycle $M$ performs at least one rewrite step – thus each cycle reduces the weight of the actual tape content with respect to the weight function $\omega$ mentioned above. Further, it is required that $M$ does not perform any rewrite steps in a tail. We use the notation $u \vdash_M^\gamma v$ to denote a cycle of $M$ that begins with the restarting configuration $q_0\alpha\|u$ and ends with the restarting configuration $q_0\alpha\|v$; the relation $\vdash_M^\gamma$ is the reflexive and transitive closure of $\vdash_M^\gamma$. The pair $RS(M) := (\Gamma^*, \vdash_M^\gamma)$ is called the reduction system induced by $M$.

A sentential form $w \in \Gamma^*$ is accepted by $M$, if there is an accepting computation which starts from the restarting configuration $q_0w\|$. By $LSF(M)$ we denote the language consisting of all sentential forms accepted by $M$; we say that $LSF(M)$ is the language of sentential forms recognized by $M$.

From the above description it is easily concluded that, starting from a configuration of the form $q_0w\|$, $M$ will execute at most $c \cdot |w|$ many cycles for some constant $c \in \mathbb{N}_+$, which implies that $LSF(M)$ is accepted by a nondeterministic Turing machine simultaneously in quadratic time and in linear space, that is, $LSF(M) \in \text{NP} \cap \text{CSL}$.

If a proper subalphabet $\Sigma$ of $\Gamma$ is fixed as an alphabet of terminal symbols (or input symbols), then the language $L_{\Gamma}(M) := LSF(M) \cap \Sigma^*$ of all input sentences accepted by $M$ is called the input language recognized by $M$. In this case the four-tuple $BS_{\Gamma}(M) := (\Sigma, \Gamma, LSF(M), RS(M))$ is called the basic syntactic system of $M$ with input alphabet $\Sigma$. Observe that $BS_{\Gamma}(M)$ contains the complete information about $L_{\Gamma}(M)$, $LSF(M)$, and $RS(M)$, and that $LSF(M)$ plays the main role among the two languages. In an obvious way one can introduce basic syntactic systems also for Chomsky grammars and various types of categorial and dependency grammars.

We emphasize the following nice properties of restarting automata, which are often used implicitly in proofs.

**Fact 2.1 (Error Preserving Property).** Let $M$ be an FRR-automaton, and let $u, v \in \Gamma^*$. If $u \vdash_M^\gamma v$ and $u \notin LSF(M)$, then $v \notin LSF(M)$, either.
Fact 2.2 (Correctness Preserving Property). Let $M$ be an FRR-automaton, and let $u,v \in \Gamma^*$. If $u \vdash_M v$ is a part of an accepting computation of $M$, then $v \in L_{SF}(M)$.

Observe that the latter property does in general not hold for input languages, as apart from the initial configuration, each restarting configuration in an accepting computation may contain some non-input symbols.

Finally we come to the notion of monotonicity. Let $C := \alpha \beta$ be a rewrite configuration of a FRR-automaton $M$, that is, a configuration in which a rewrite instruction is to be applied. Then $|\beta|$ is called the right distance of $C$, which is denoted by $D_r(C)$. A sequence of rewrite configurations $S = (C_1, C_2, \ldots, C_n)$ is called monotone if $D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n)$.

Let $j$ be a positive integer. We say that a sequence of rewrite configurations $S = (C_1, C_2, \ldots, C_n)$ is $j$-monotone if there is a partition of $S$ into $j$ subsequences

$$S_i = (C_{i,1}, C_{i,2}, \ldots, C_{i,p_i}), \ldots, S_j = (C_{j,1}, C_{j,2}, \ldots, C_{j,p_j})$$

such that each $S_i$, $1 \leq i \leq j$, is monotone. Observe that it is not required that the subsequences $S_1, \ldots, S_j$ follow sequentially one after another in the original sequence. Instead they are in general all scattered throughout the original sequence. Hence, a sequence of rewrite configurations $(C_1, C_2, \ldots, C_n)$ is not $j$-monotone if and only if there exist indices $1 \leq i_1 < i_2 < \cdots < i_j+1 \leq n$ such that $D_r(C_{i_1}) < D_r(C_{i_2}) < \cdots < D_r(C_{i_{j+1}})$.

A computation of an FRR-automaton $M$ is called $j$-monotone if the corresponding sequence of rewrite configurations is $j$-monotone. A computation is $j$-rewriting if none of its cycles contains more than $j$ rewrite steps. Finally, a computation is $j$-constrained if it is both $j$-rewriting and $j$-monotone.

Notation. For any class $A$ of automata and any index $Y \in \{I, SF\}$, $L_Y(A)$ will denote the class of $Y$-languages recognizable by automata from $A$, and $BS_I(A)$ will denote the class of basic syntactic systems determined by $A$. By $\text{(D)CFL}$ we denote the class of (deterministic) context-free languages, and by $\subset$ we denote the proper subset relation. Sometimes we will use regular expressions instead of the corresponding regular languages.

3 Constraints of word-order

In this section we introduce some constraints which will be used for characterizations of word-order complexity and word-order freedom. For us the degree of word-order freedom means a certain degree of robustness against permutations of sentential forms and their reductions, while the degree of constrainability serves as a synonym for the degree of word-order complexity. In the following we will restrict our attention mainly to languages of sentential forms. First we introduce an infinite hierarchy of classes of languages of sentential forms based on the notion of constrainability.
Definition 3.1 For an FRR-automaton $M$ over $\Gamma$, and an integer $i \in \mathbb{N}_+$,

$$L_{SF}(M, i) := \{ w \in L_{SF}(M) \mid M \text{ accepts } w \text{ by an } i\text{-constrained computation} \}.$$

In addition, if an input alphabet $\Sigma \subset \Gamma$ is fixed for $M$, then

$$L_{I}(M, i) := \{ w \in L_{I}(M) \mid M \text{ accepts } w \text{ by an } i\text{-constrained computation} \}.$$

For each FRR-automaton $M$, each input alphabet $\Sigma$, each $Y \in \{I, SF\}$, and each $i \geq 1$, $L_{Y}(M, i) \subseteq L_{Y}(M, i+1)$, as each $i$-constrained computation is also $(i+1)$-constrained.

Definition 3.2 Let $i \in \mathbb{N}_+$, and let $\mathcal{A}$ be a type of restarting automaton. Then

$$L_{SF}(i, \mathcal{A}) := \bigcup_{j=1}^{i} \{ L_{SF}(M, j) \mid M \text{ is an } \mathcal{A}\text{-automaton} \}.$$ 

Further, by

$$L_{I}(i, \mathcal{A}) := \bigcup_{j=1}^{i} \{ L_{I}(M, j) \mid M \text{ is an } \mathcal{A}\text{-automaton} \}$$

we denote the corresponding class of languages obtained when considering $\mathcal{A}$-automata with designated input alphabets.

The above definition trivially ensures the fact that, for all $i \geq 1$,

$$L_{SF}(i, \text{FRR}) \subseteq L_{SF}(i+1, \text{FRR}) \text{ and } L_{I}(i, \text{FRR}) \subseteq L_{I}(i+1, \text{FRR})$$

hold.

Theorem 3.1 DCFL $\subset L_{SF}(1, \text{FRR}) \subset$ CFL $= L_{I}(1, \text{FRR})$.

Proof. In [12] the shrinking restarting automaton is introduced (see also [13]). It differs from the standard restarting automaton in that rewrite steps of the form $u \rightarrow v$ are not necessarily length-reducing, but that they are only required to be weight-reducing with respect to some weight function. According to [9] CFL coincides with the class of languages that are accepted by monotone restarting automata with auxiliary symbols (that is, monotone RRWW-automata in the notation of [16]). In fact, CFL coincides with the class of languages that are accepted by monotone shrinking RRWW-automata [12]. However, it is easily seen that monotone shrinking RRWW-automata correspond to FRR-automata with designated input alphabets executing only 1-constrained computations. This yields the equality $\text{CFL} = \mathcal{L}(\text{mon-sRRWW}) = \mathcal{L}_{I}(1, \text{FRR})$.

Given an FRR-automaton $M$ without a designated input alphabet, one can easily construct a monotone shrinking RRWW-automata that simulates the 1-constrained computations of $M$, which gives the inclusion $L_{SF}(1, \text{FRR}) \subseteq \text{CFL}$. On the other hand, consider the context-free language $L_{ab} := \{ a^n b^n \mid n \geq 0 \} \cup \{ a^n b^3n \mid n \geq 0 \}$. Based
on the Error Preserving Property it can be shown easily that \( L_{ab} \not\subseteq L_{SF}(1, FRR) \). Hence, we obtain the proper inclusion \( L_{SF}(1, FRR) \subset CFL \).

From [9] we know that DCFL coincides with the class of languages that are accepted by deterministic monotone restarting automata without auxiliary symbols (that is, det-mon-RR-automata in the notation of [16]). Clearly each automaton of this form can be simulated by an FRR-automaton executing only 1-constrained computations, which implies that DCFL \( \subseteq L_{SF}(1, FRR) \). Further, for the language \( L_{ac} := \{ a^n b^m \mid 1 \leq n \leq m \leq 2n \} \), which is not deterministic context-free, there exists an FRR-automaton such that \( L_{ac} \) is accepted by the 1-constrained computations of that automaton. Thus, we obtain the proper inclusion \( DCFL \subset L_{SF}(1, FRR) \).

Next we introduce a restricted type of FRR-automaton called MRR-automaton. The transition relation of an MRR-automaton \( M \) is given through a finite sequence of so-called meta-instructions of the form

\[
(E_1, u_1 \rightarrow v_1, E_2, u_2 \rightarrow v_2, E_3, \ldots, E_i, u_i \rightarrow v_i, E_{i+1}),
\]

where \( E_1, \ldots, E_{i+1} \) are regular expressions, and for each \( n = 1, \ldots, i \), \( u_n, v_n \in \Gamma^* \) are strings satisfying \( \omega(u_n) < \omega(v_n) \), where \( \omega : \Gamma \rightarrow \mathbb{N}_+ \) is a weight function associated with \( M \). The rules \( u_1 \rightarrow v_1, u_2 \rightarrow v_2, \ldots, u_i \rightarrow v_i \) embody rewrite steps of \( M \). On trying to execute this meta-instruction, \( M \) will get stuck (and so reject) starting from the configuration \( q_0 q w \$ \), if \( w \) does not admit a factorization of the form \( w = w_1 u_1 w_2 u_2 \cdots w_i u_i w_{i+1} \) such that \( q w_1 \in L(E_1), w_2 \in L(E_2), \ldots, w_{i+1} \in L(E_{i+1}) \), where \( L(E_n) \) denotes the language described by the regular expression \( E_n \). On the other hand, if \( w \) does have factorizations of this form, then one such factorization is chosen nondeterministically, and \( q_0 q w \$ \) is transformed into \( q_0 q w_1 v_1 w_2 v_2 w_3 \cdots v_i w_{i+1} \$ \). In order to describe the tails of accepting computations of \( M \) (during which \( M \) cannot apply any rewrite operations at all), we use meta-instructions of the form \( (q \cdot E \cdot \$, Accept \) ), which accepts the sentences from the regular language \( L(E) \).

For each MRR-automaton \( M \), there exists a number \( j_M \) such that no cycle of any computation of \( M \) contains more than \( j_M \) applications of rewrite steps, that is, each computation of \( M \) is \( j_M \)-rewriting. Thus, MRR-automata have limited rewriting only. In this sense MRR-automata are weaker than FRR-automata.

**Example 3.1** We consider the language \( L_\infty := \{ (a^n b^n)^m \mid n, m \geq 0 \} \). It is easily seen that the following deterministic FRR-automaton \( M_\infty \) with initial state \( q_0 \) and tape alphabet \( \Gamma := \{ a, b \} \) accepts this language:

<table>
<thead>
<tr>
<th>( \delta(q_0, a$) )</th>
<th>( \delta(q_0, a a a) )</th>
<th>( \delta(q_1, a a a) )</th>
<th>( \delta(q_1, a a b) )</th>
<th>( \delta(q_1, b a a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>( q_0 ), MVR</td>
<td>( q_0 ), MVR</td>
<td>( q_0 ), MVR</td>
<td>( q_0 ), MVR</td>
</tr>
<tr>
<td>( q_0 ), MVR</td>
<td>( q_0 ), MVR</td>
<td>( q_0 ), MVR</td>
<td>( q_0 ), MVR</td>
<td>( q_0 ), MVR</td>
</tr>
<tr>
<td>( q_1 ), MVR</td>
<td>( q_1 ), $</td>
<td>( q_0 ), qab</td>
<td>( q_2 ), qab</td>
<td>( q_2 ), $</td>
</tr>
<tr>
<td>( q_1 ), MVR</td>
<td>( q_2 ), $</td>
<td>( q_2 ), $</td>
<td>( q_2 ), $</td>
<td>( q_2 ), $</td>
</tr>
<tr>
<td>( q_2 ), $</td>
<td>Accept</td>
<td>Accept</td>
<td>Accept</td>
<td>Accept</td>
</tr>
</tbody>
</table>
As each rewrite instruction of \( M_\infty \) simply deletes a factor of the form \( ab \), it is clear that \( M_\infty \) is compatible with an arbitrary weight function \( \omega : \Gamma \to \mathbb{N}_+ \). Given the string \( w = (a^3b^3)^4 \) as input, \( M_\infty \) will execute the following computation:

\[
\begin{align*}
& q_0 \alpha a^3b^3(a^3b^3)^3 \beta & 
\vdash^* & q_0a^2q_0abbb(a^3b^3)^3 \beta \\
& & \vdash_{M_\infty} q_0a^2bq_1b(a^3b^3)^3 \beta \\
& & \vdash^*_{M_\infty} q_0^2a^2q_0abbb(a^3b^3)^2 \beta \\
& & \vdash_{M_\infty} q_0a^2b^2a^2bq_1b(a^3b^3)^2 \beta \\
& & \vdash^*_{M_\infty} q(a^2b^2)^2a^2q_0abba^3b^3 \beta \\
& & \vdash_{M_\infty} q(a^2b^2)^2a^2bq_1ba^3b^3 \beta \\
& & \vdash^*_{M_\infty} q(a^2b^2)^3a^2q_0abbb \beta \\
& & \vdash_{M_\infty} q(a^2b^2)^3a^2bq_1b \beta \\
& & \vdash^*_{M_\infty} q_0a^2aabb(a^2b^2)^3 \beta \\
& & \vdash_{M_\infty} q_0a^2q_0abbb(a^2b^2)^3 \beta \\
& & \vdash^*_{M_\infty} qabq_1aabbb(a^2b^2)^2 \beta \\
& & \vdash_{M_\infty} qabaq_0abba^2b^2 \beta \\
& & \vdash^*_{M_\infty} qababq_1aabb \beta \\
& & \vdash_{M_\infty} qababq_0aabbb \beta \\
& & \vdash^*_{M_\infty} qabababq_1 \beta \\
& & \vdash_{M_\infty} qq_2ababab \beta \\
& & \vdash^*_{M_\infty} qq_2 \vdash_{M_\infty} \text{Accept}.
\end{align*}
\]

On the other hand, if \( M' \) is an MRR-automaton on \( \Gamma \), then \( M' \) has no accepting computation for an input of the form \( w := (a^nb^n)^{j_{M'}+1} \), where \( n \) is sufficiently large. Indeed, as \( M' \) can execute at most \( j_{M'} \) rewrite steps per cycle, the first cycle of \( M' \) in an accepting computation on input \( w \) will transform the string \( w \) into a string not belonging to the language \( L_\infty \), thus violating the Correctness Preserving Property.

This example has the following consequence.

**Corollary 3.1** \( \mathcal{L}_{SF}(\text{MRR}) \subseteq \mathcal{L}_{SF}(\text{FRR}) \).

For the FRR-automaton \( M_\infty \) of Example 3.1 we see that

\[
L_{SF}(M_\infty, i) = \{ (a^nb^n)^m \mid n \geq 0, m \leq i \},
\]

which shows that the languages \( L_{SF}(M_\infty, i) \) \( (i \geq 1) \) form an infinite strictly ascending sequence approximating the language \( L_{SF}(M_\infty) = L_\infty \). Again the language \( \{ (a^nb^n)^m \mid n \geq 0, m \leq i \} \) cannot be accepted by \( (i - 1) \)-constrained computations of any FRR-automaton. Hence, we have the following infinite hierarchy.

**Corollary 3.2** For all \( i \geq 1 \), \( \mathcal{L}_{SF}(i, \text{FRR}) \subseteq \mathcal{L}_{SF}(i + 1, \text{FRR}) \).
Example 3.2 Let \( M_1 \) be the MRR-automaton with tape alphabet \( \{a,b\} \) that is given through the following set of meta-instructions:

\[
\begin{align*}
(1) \quad & (\mathfrak{q} \cdot a^+, abb \to b, b^+ \cdot \$), \\
(2) \quad & (\mathfrak{q} \cdot a^+, abb \to b, b^* \cdot a^+, abb \to b, b^+ \cdot \$), \\
(3) \quad & (\mathfrak{q} \cdot (abab) \cdot \$, Accept).
\end{align*}
\]

It is easy to see that \( L_{SF}(M_1,1) = \{a^n b^n \mid n > 0\} \cup \{abab\} \), as in 1-constrained computations \( M_1 \) can only use meta-instructions (1) and (3), while \( L_{SF}(M_1,2) = L_{SF}(M_1,1) \cup \{a^n b^n a^n b^n \mid n > 0\} \). Hence, \( L_{SF}(M_1) = L_{SF}(M_1,2) \supset L_{SF}(M_1,1) \).

Observe that \( L_{SF}(M_1) = L_{SF}(M_1,2) = \{a^n b^n, a^n b^n a^n b^n \mid n > 0\} \) is not context-free, and so \( L_{SF}(M_1) \notin L_{SF}(1,\text{FRR}) \) by Theorem 3.1.

Example 3.3 Let \( M_2 \) be the MRR-automaton with the same tape alphabet as \( M_1 \) that is given through the meta-instructions (1) to (3) of \( M_1 \) and the following two instructions:

\[
\begin{align*}
(4) \quad & (\mathfrak{q} \cdot a^+, abb \to b, b^* \cdot ab \cdot \$), \\
(5) \quad & (\mathfrak{q} \cdot a^+, abb \to b, b^+ \cdot \$).
\end{align*}
\]

In 1-constrained computations \( M_2 \) can use the meta-instructions (1) and (3) to (5). It follows easily that \( L_{SF}(M_2,1) = \{a^n b^n, a b a^n b^n, a^n b^n a n b a b \mid n > 0\} \), while \( L_{SF}(M_2,2) = \{a^n b^n a^n b a b \mid n > 0, m \geq 0\} = L_{SF}(M_2) \).

In contrast to \( L_{SF}(M_1) \), the language \( L_{SF}(M_2) \) is context-free, and \( L_{SF}(M_2) \) can even be accepted by an MRR-automaton \( M' \) that executes only 1-constrained computations, that is, \( L_{SF}(M_2) \in L_{SF}(1,\text{FRR}) \). \( M' \) is given through the following meta-instructions:

\[
\begin{align*}
(1) \quad & (\mathfrak{q} \cdot a^+, abb \to b, b^* \cdot \$), \\
(2) \quad & (\mathfrak{q} \cdot a^+, abb \to b, b^* \cdot a^+ \cdot b^+ \cdot \$), \\
(3) \quad & (\mathfrak{q} \cdot a^+, abb \to b, b^+ \cdot \$), \\
(4) \quad & (\mathfrak{q} \cdot a^+, abb \to b, b^+ \cdot \$), \\
(5) \quad & (\mathfrak{q} \cdot a^+, abb \to b, b^+ \cdot \$).
\end{align*}
\]

Below we will use the following families of sample languages, where \( j \in \mathbb{N}_+ \), \( \Gamma_j := \{a,b,c_0,c_1, \ldots, c_{j-1}\} \), and \( \Delta_j := \{a_1, a_2, \ldots, a_j\} \):

\[
\begin{align*}
LC_j & := \{ c_0 w c_1 w \cdots c_{j-2} w c_{j-1} w \mid w \in \{a,b\}^* \} \subset \Gamma_j, \\
LE_j & := \{ w \in \{a_1, \ldots, a_j\}^* \mid | w |_{a_1} = | w |_{a_2} = \cdots = | w |_{a_j} \} \subset \Delta_j.
\end{align*}
\]

Let us remark that the languages \( LC_j (j \geq 1) \) represent those languages in which valences are modelled through fixed reduction positions (with an increasing number of such positions), and the languages \( LE_j (j \geq 1) \) represent those languages in which valences are modelled by \( j \)-tuples of symbols in varying positions, independent of the word-order.

Example 3.4 Let \( j \geq 1 \), and let \( MC_j \) be the MRR-automaton with tape alphabet \( \Gamma_j \) that is given through the following sequence of meta-instructions, where \( \Sigma_0 := \{a,b\} \):

\[
\begin{align*}
(1) \quad & (\mathfrak{q} \cdot a \to \lambda, \Sigma_0^* \cdot c_1, a \to \lambda, \Sigma_0^* \cdot c_2, \ldots, \Sigma_0^* \cdot c_{j-1}, a \to \lambda, \Sigma_0^* \cdot \$), \\
(2) \quad & (\mathfrak{q} \cdot b \to \lambda, \Sigma_0^* \cdot c_1, b \to \lambda, \Sigma_0^* \cdot c_2, \ldots, \Sigma_0^* \cdot c_{j-1}, b \to \lambda, \Sigma_0^* \cdot \$), \\
(3) \quad & (\mathfrak{q} \cdot \$ \text{Accept}).
\end{align*}
\]
It follows immediately that \( L_{SF}(MC_j) = LC_j \) holds, and that all computations of \( MC_j \) are \( j \)-constrained. On the other hand, it is easily seen that \( LC_{j+1} \not\subseteq L_{SF}(j, FRR) \), as for a sentence \( x \in LC_{j+1} \), \( j \) rewrite operations per cycle do in general not suffice to transform \( x \) into another string that still belongs to \( LC_{j+1} \). Hence, \( LC_{j+1} \) does not coincide with the language of sentential forms that are accepted by \( j \)-constrained computations of any \( FRR \)-automaton.

However, for each \( j \geq 1 \), there exists an \( MRR \)-automaton \( \overline{M}_j \) with input alphabet \( \Gamma_j \) and tape alphabet \( \Omega_j := \Gamma_j \cup D \), where \( D := \{ d_a, d_b \} \), such that \( L_I(\overline{M}_j, 2) = LC_j \). The automaton \( \overline{M}_j \) is given through the following meta-instructions, where \( u \in \Sigma_0^* \):

\[
\begin{align*}
(1) & \quad (\# c_0, u \rightarrow \lambda, \Sigma_0^* \cdot c_1 \cdot D^*, u \rightarrow d_u, \Sigma_0^* \cdot c_2 \cdot \cdots \cdot c_{j-1} \cdot \Sigma_0^* \cdot \#), \\
(2) & \quad (\# c_0 c_1, d_u \rightarrow \lambda, D^* \cdot c_2 \cdot D^*, u \rightarrow d_u, \Sigma_0^* \cdot c_3 \cdot \cdots \cdot c_{j-1} \cdot \Sigma_0^* \cdot \#), \\
\ldots & \\
(j-2) & \quad (\# c_0 \cdots c_{j-3}, d_u \rightarrow \lambda, D^* \cdot c_{j-2} \cdot D^*, u \rightarrow d_u, \Sigma_0^* \cdot c_{j-1} \cdot \Sigma_0^* \cdot \#), \\
(j-1) & \quad (\# c_0 c_1 \cdots c_{j-2}, d_u \rightarrow \lambda, D^* \cdot c_{j-1}, u \rightarrow \lambda, \Sigma_0^* \cdot \#), \\
(j) & \quad (\# c_0 c_1 \cdots c_{j-1} \# \text{Accept}).
\end{align*}
\]

Here we take the weight function \( \omega(x) := 2 \) (\( x \in \Gamma_j \)) and \( \omega(d) := 1 \) (\( d \in D \)).

Given an input \( c_0 w_1 c_1 w_2 \cdots c_{j-2} w_{j-1} c_{j-1} w_j \) with \( w_1, \ldots, w_j \in \Sigma_0^* \), \( \overline{M}_j \) gradually compares the neighbouring factors \( w_i \) and \( w_{i+1} \) (\( 1 \leq i \leq j-1 \)) from left to right. First meta-instructions of type (1) are used repeatedly to compare \( w_1 \) to \( w_2 \), thereby deleting \( w_1 \) letter by letter from left to right, and encoding each letter \( u \) of \( w_2 \) by the letter \( d_u \). Then meta-instructions of type (2) are used to compare \( w_2 \) to \( w_3 \), thereby deleting (the encoded version of) \( w_2 \) letter by letter from left to right, and encoding each letter \( u \) of \( w_3 \) by the letter \( d_u \). This continues for indices \( 3, 4, \ldots, j-2 \) until finally meta-instructions of type \( (j-1) \) are used repeatedly to compare (the encoded version of) \( w_{j-1} \) and \( w_j \), thereby deleting both syllables from left to right. It follows that each computation of \( \overline{M}_j \) is \( 2 \)-constrained, that is, \( LC_j \subseteq L_I(2, FRR) \).

Thus, when a language is interpreted as the input language of an \( FRR \)-automaton, then it may be lower in the hierarchy as when it is considered as the language of sentential forms of an \( FRR \)-automaton. The reason for this phenomenon is the fact that the Correctness Preserving Property does in general not hold for input languages. The following theorem summarizes these observations.

**Theorem 3.2**

(a) \( L_{SF}(FRR) \subset L_I(FRR) \), and

(b) \( L_{SF}(i, FRR) \subset L_I(i, FRR) \) for all \( i \in \mathbb{N}_+ \).

**Proof.** For \( i \geq 2 \) the proper inclusion in (b) is an immediate consequence of Example 3.4, while for \( i = 1 \) this is simply a part of the statement of Theorem 3.1. For proving (a) consider the language \( L'_\infty := \{ (a^n b^n)^n \mid n \geq 1 \} \).

**Claim 1.** \( L'_\infty \in L_I(FRR) \).

**Proof.** We describe an \( FRR \)-automaton \( M' \) for the language \( L'_\infty \). This automaton will use tape alphabet \( \Gamma := \Sigma_0 \cup \{ A, B \} \), input alphabet \( \Sigma_0 \), and the weight function \( \omega \).
Degrees of Free Word-Order

that is defined through $\omega(a) := \omega(b) := 2$ and $\omega(A) := \omega(B) := 1$. Given an input of the form $w := a^{m_1}b^{n_1}a^{m_2}b^{n_2} \cdots a^{m_t}b^{n_t}$ ($t > 0, n_i, m_i > 0, i = 1, \ldots, t$), $M'$ will proceed as follows.

Each restarting configuration that $M'$ reaches during its computation on input $w$ will be of the form

$$C_j := q_0\text{w}a^{m_1}B^{n_1} \cdots A^{m_j}B^{n_j}a^{m_{j+1}}b^{n_{j+1}} \cdots a^{m_t}b^{n_t}.$$

where the initial configuration $q_0\text{w}$ is just the special case $C_0$. $M'$ will accept starting from $C_j$, if the tape content of $C_j$ is from the regular language

$$q \cdot (AB)^* \cdot ab \cdot \text{w}.$$

If the tape content is not of this form, then $M'$ executes a cycle that comprises the following operations when starting from $C_j$:

- In each block of the form $A^{m_i}B^{n_i} (1 \leq i \leq j)$, a factor $AB$ is deleted. If $m_i - j = 1$ or $n_i - j = 1$, then $M'$ halts and rejects.
- The block $a^{m_{j+1}}b^{n_{j+1}}$ is rewritten into $A^{m_{j+1}}B^{n_{j+1}}$. If $m_{j+1} - j = 1$ or $n_{j+1} - j = 1$, then $M'$ halts and rejects.
- In each block of the form $a^{m_{j+1}}b^{n_{j+1}} (2 \leq i \leq t - j)$, a factor $ab$ is deleted. Again, if $m_{j+i} - j = 1$ of $n_{j+i} - j = 1$, then $M'$ halts and rejects.
- Upon reaching the right delimiter $\$, $M'$ restarts.

If $M'$ does not reject while performing these operations, then it reaches the restarting configuration $C_{j+1}$. Hence, the computation of $M'$ on input $w$ has the form

$$C_0 \xrightarrow{c}_{M'} C_1 \xrightarrow{c}_{M'} \cdots \xrightarrow{c}_{M'} C_k,$$

and either $M'$ halts and rejects starting from $C_k$, or $k = t - 1$ and $C_k = q_0\text{w}(AB)^{t-1}ab\$, in which case $M'$ halts and accepts. It follows that $L_T(M') = L'_\infty$. \hfill $\Box$

Claim 2. $L'_\infty \not\subset L_S(FRR)$.

Proof. Assume that $L'_\infty = L_S(FRR)$ for some FRR-automaton $M$. Then $M$ has tape alphabet $\Sigma_0$ only. Given an input of the form $(a^n b^m)^n$ for sufficiently large $n$, $M$ cannot accept in a tail computation, that is, starting from the initial configuration $q_0\text{w}(a^n b^m)^n\$, an accepting computation of $M$ will begin with a cycle that leads to a restarting configuration $q_0\text{w}\$. As $M$ must satisfy the Correctness Preserving Property, we see that $u \in L'_\infty$ holds. Further, as each rewrite step of $M$ reduces the weight of the tape content with respect to an appropriate weight function, we see that $u$ must be of the form $u = (a^m b^m)^m$ for some integer $m < n$. Thus, in the cycle above $M$ deletes $n - m > 0$ blocks of the form $a^n b^m$, and it rewrites all remaining blocks of the form $a^n b^m$ into $a^m b^m$. If $n$ is sufficiently large, then $M$ cannot ensure that the deleted blocks are all of the correct length. Hence, there exists an integer $j > 0$ and an index $n_j \in \{1, \ldots, n\}$ such that $M$ will also execute the cycle
(a^n b^n)^{n_1} a^{n+j} b^{n+j} (a^n b^n)^{n-n_1-1} \equiv_M (a^m b^m)^{n}, which contradicts the Error Preserving Property.

This completes the proof of Theorem 3.2.

Next we introduce some notions in order to formalize the degree of word-order freedom of languages.

**Definition 3.3** (a) Two strings $u, v \in \Gamma^*$ are called Parikh equivalent, denoted by $u \equiv v$, if $|u|_a = |v|_a$ holds for each $a \in \Gamma$.

(b) Let $M$ be an FRR-automaton, and let $u, v \in L_{SF}(M)$. We say that $u$ is $M$-transformable into $v$, denoted by $u \equiv_{\Rightarrow}^M v$, if $u \equiv v$ holds, and for each string $u_1$ satisfying $u \vdash_{M}^* u_1$ and $|u| > |u_1|$, there exists a string $v_1$ such that $v \vdash_{M}^* v_1$ and $u_1 \equiv v_1$.

(c) An FRR-automaton $M$ is called reduction-preserving if, for each $i > 0$ and each $w \in L_S(M, i+1)$, there exists a string $v \in L_{SF}(M, i)$ such that $w \equiv_{\Rightarrow}^M v$.

If $M$ is reduction-preserving, then for each $i > 0$ and each string $w$ that is accepted by $M$ through an $(i+1)$-constrained computation, there exists a string $v$ that is accepted by $M$ through an $i$-constrained computation such that $w$ and $v$ are Parikh equivalent, and even more, each string that is shorter than $w$ and that can be reached from $w$ by a reduction modulo $M$ is also Parikh equivalent to some string that can be reached from $v$ by a number of reductions modulo $M$. Intuitively, this means that by commuting $w$ we obtain a string $v$ that is of a lower degree of word-order complexity, but that admits essentially the same reductions (modulo commutation).

**Example 3.5** The MRR-automaton $M_1$ of Example 3.2 is reduction-preserving. Indeed, let $n \geq 2$ and $u := a^n b^n a^n b^n \in L_{SF}(M_1, 2) \setminus L_{SF}(M_1, 1)$. Choose $v := a^{2n} b^{2n} \in L_{SF}(M_1, 1)$. Then $u \equiv_{\Rightarrow}^M v$, as $u \equiv v$ and $u \vdash_{M_1}^* u_1$ implies that $u_1 = a^{n-1} b^{n-1} a^{n-1} b^{n-1}$, and so $v_1 := a^{2n-2} b^{2n-2} \in L_{SF}(M_1, 1)$ satisfies the conditions $v \vdash_{M_1}^* v_1$ and $v_1 \equiv v_1$.

For an FRR-automaton $M$, if $L_{SF}(M, i) = L_{SF}(M, 1)$ holds for all $i \geq 1$, then $M$ is trivially reduction-preserving. The following notion will be used for reduction-preserving FRR-automata to measure their distance from this trivial case.

**Definition 3.4** A reduction-preserving FRR-automaton $M$ is called $j$-scalable for some $j \in \mathbb{N}$, if, for each $i = 1, \ldots, j - 1$, $L_{SF}(M, i+1) \notin L_{SF}(i, FRR)$. A language $L_{SF}$ of sentential forms is $j$-scalable, if there exists a $j$-scalable FRR-automaton $M$ such that $L_{SF} = L_{SF}(M)$.

Thus, if an FRR-automaton $M$ is $j$-scalable, then, for all $1 \leq i \leq j - 1$, the language $L_{SF}(M, i+1)$ of sentential forms that $M$ accepts through $(i+1)$-constrained computations is not accepted by any FRR-automaton performing only $i$-constrained computations. Of course, each reduction-preserving FRR-automaton is 1-scalable.
Claim 1. If an FRR-automaton $M$ is $j$-scalable for some integer $j > 1$, then $M$ is $(j - 1)$-scalable as well.

We now illustrate the above concepts by a few examples.

Example 3.6 As seen above the MRR-automaton $M_1$ of Example 3.2 is reduction-preserving. Further, the language $L_{SF}(M_1, 2) = \{ a^n b^n, a^n b^n a^n b^n \mid n > 0 \}$ does not coincide with the language of sentential forms that an FRR-automaton can accept by 1-constrained computations. Thus, $M_1$ is 2-scalable. However, it is not 3-scalable, as $L_{SF}(M_1) = L_{SF}(M_1, 2)$. On the other hand, the MRR-automaton $M_2$ of Example 3.3 is only 1-scalable, but it is not 2-scalable, as the language $L_{SF}(M_2, 2)$ is also accepted by the MRR-automaton $M'$ performing only 1-constrained computations.

Example 3.7 The language $LC_2$ is not accepted by any reduction-preserving FRR-automaton. Indeed, assume that $M$ is an FRR-automaton for $LC_2$ with a read/write window of size $k$. Consider the word $w_n := c_0 a^n b^k n c_1 a^n b^k n \in LC_2$, where $n$ is sufficiently large. If $w_n$ is Parikh equivalent to a word of the form $c_0 a^m c_1 a^m \in LC_2$, then $u$ must contain at least one occurrence of the symbol $a$. This implies that $m \leq n$, which in turn means that $|u| \geq k + 1$. Through any rewrite step $M$ can delete at most $k$ symbols, and so it cannot delete a copy of the syllable $u$ from both factors each in a single rewrite step. Hence, we see that $c_0 a^m c_1 a^m$ cannot be accepted by $M$ through a 1-constrained computation. Thus, $LC_2$ is not accepted by any reduction-preserving FRR-automaton, and therewith $LC_2$ is not $j$-scalable for any $j \geq 1$. The same argument applies to each of the languages $LC_j$ ($j \geq 3$).

Finally we turn to the family of languages $LE_j$ ($j \geq 1$).

**Proposition 3.2** For $j \geq 2$, $LE_j$ is $(j - 1)$-scalable, but it is not $j$-scalable.

**Proof.** Let $j \geq 2$, and let $M_j$ be the (deterministic) MRR-automaton with tape alphabet $\Delta_j = \{a_1, \ldots, a_j\}$ that is defined by the following meta-instructions, where $\pi$ ranges over the set of permutations of the index set $\{1, 2, \ldots, j\}$, and $i$ ranges over the set of indices $3, \ldots, j - 1$:

1. $\pi \cdot \lambda$ : $a_1^* a_{\pi(1)}^* a_{\pi(2)}^* \lambda, a_{\pi(1)}^* a_{\pi(2)}^* \lambda, \ldots, a_{\pi(j)}^* \lambda, \lambda, \lambda, \Delta_j^* \cdot \lambda$,
2. $\pi \cdot \lambda$ : $a_2 a_1 \lambda, a_3 \lambda, \lambda, a_5 a_1 a_4 \lambda, \lambda, a_1^* a_5^* a_1^*$,
3. $\lambda$ : $a_1^* a_1^* a_{i+1} \lambda, \lambda, (a_{i+1} \cdots a_j)^* \lambda$.

**Claim 1.** $L_{SF}(M_j) = LE_j$, and every computation of $M_j$ is $(j - 1)$-constrained.

**Proof.** If the tape content is of the form

$$w(i) := a_n^2 a_1^{m_2} a_3^{m_3} a_4^{m_4} \cdots a_i^{m_i} a_1^{n-1} (a_{i+1} \cdots a_j)^n$$
for some integers \(n, m_1, \ldots, m_{i-1} > 0\), then meta-instruction \((2, i)\) is applicable. It leads to acceptance if and only if \(w(i) \in LE_j\), and the corresponding accepting computation is obviously \(i\)-constrained. If the tape content is not of the form above, then \(M_j\) can only apply a meta-instructions of the form \((1, \pi)\), and hence, in each cycle \(M_j\) executes exactly \(j - 1\) delete operations. If we arrange the rewrite (delete) configurations of a computation of \(M_j\) into \(j - 1\) subsequences in such a way that the \(i\)-th delete configuration of each cycle is put into the \(i\)-th subsequence for all \(i = 1, \ldots, j - 1\), then it can be verified that the \(j - 1\) subsequences obtained are all monotone. It is not hard to see that \(M_j\) recognizes \(LE_j\).

It is easily seen that \(M_j\) is reduction-preserving. On the other hand, we have the following result.

**Claim 2.** For all \(i \in \{2, \ldots, j - 1\}\), \(L_{SF}(M_j, i) \notin L_{SF}(i - 1, \text{FRR})\).

**Proof.** Consider an input of the form \(w(i)\) such that \(n = m_1 + \ldots + m_{i-1}\), and all integers \(m_1, \ldots, m_{i-1}\) are sufficiently large. In order to transform \(w(i)\) into a shorter sentence also belonging to \(L_{SF}(M_j, i)\), at least one occurrence of each letter \(a_1, a_2, \ldots, a_j\) must be deleted. If the integers \(m_2, \ldots, m_{i-1}\) are larger than the size of the window of the automaton considered, then in order to delete at least one occurrence of each letter, the automaton must perform at least \(i\) delete operations. \(\Box\)

This completes the proof of Proposition 3.2. \(\Box\)

The next theorem, which is a consequence of the above proposition, shows that there exists an infinite sequence of classes of languages with an increasing degree of word-order freedom and word-order complexity as well.

**Theorem 3.3** For all \(i, j \in \mathbb{N}_+, i < j\),

\(a\) \(L_{SF}((j + 1)\text{-scal-FRR}) \subset L_{SF}(j\text{-scal-FRR})\).
\(b\) \(L_{SF}(i, j\text{-scal-FRR}) \subset L_{SF}(i + 1, j\text{-scal-FRR})\).

**Proof.** The inclusions in \(a\) follows from Proposition 3.1, while Proposition 3.2 shows that these inclusions are proper. Claim 2 of the proof of Proposition 3.2 implies that the inclusions in \(b\) are proper. \(\Box\)

### 4 Conclusions

We consider the notion of \(j\)-constrainability as a measure for the complexity of the word-order of (individual syntactic phenomena of) natural languages, while the notion of \(j\)-scalability serves as a measure for the magnitude of the word-order freedom of languages. We have seen that the word-order complexity can serve as a parameter for restarting automata. This fact is valuable particularly for the construction of syntactic analyzers for (complex) free word-order languages. Moreover, we have seen that the word-order complexity can be caused by phenomena that are based on fixed
syntactic positions as well as by phenomena that are based on flexible (free) syntactic positions.

Each context-free language is the input language accepted by 1-constrained computations of an FRR-automaton, while the languages of sentential forms accepted by 1-constrained computations of FRR-automata form a proper subclass of CFL (Theorem 3.1). On the other hand, the classes of languages of sentential forms that are accepted by \( i \)-constrained computations of FRR-automata form an infinite strict hierarchy with respect to the parameter \( i \) (Corollary 3.2). Do we obtain a corresponding hierarchy within CFL? Or is there an upper limit on the degree of constrainability for context-free languages of sentential forms of FRR-automata?

For example, consider the context-free language \( L_{\text{pal}} := \{ ww^R \mid w \in \{a, b\}^* \} \). It is easily seen that this language is accepted by 2-constrained computations of an FRR-automaton that, in each cycle, simply compares and removes the first symbol and the last symbol of the actual tape content. On the other hand, it appears highly unlikely that the language \( L_{\text{pal}} \) coincides with the language of sentential forms that are accepted by 1-constrained computations of an FRR-automaton.

Hence, we see that there seems to be context-free languages that belong to the difference set \( L_{SF}(2, \text{FRR}) \setminus L_{SF}(1, \text{FRR}) \). Are there also context-free languages in the set \( L_{SF}(i + 1, \text{FRR}) \setminus L_{SF}(i, \text{FRR}) \) for larger values of \( i \)? Are there actually context-free languages that are not accepted as languages of sentential forms by any FRR-automaton at all?

For the future we also plan to characterize the basic syntactic power of tree-adjoining grammars [10] using techniques from [2], various types of categorial grammars (see, e.g., [1]) and of other tools [5, 6] based on the notions outlined in this paper. This seems to be promising, in particular for grammars based on topological constraints [5], because of the similar (in fact topological) type of constraints studied here.
References


