BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

ON PARALLEL PROCESSING IN FORMAL MODELS: JUMPING AUTOMATA AND NORMAL FORMS

O PARALELNÍM ZPRACOVÁNÍ VE FORMÁLNÍCH MODELECH

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR **Ing. RADIM KOCMAN**
AUTOR PRÁCE

SUPERVISOR **prof. RNDr. ALEXANDER MEDUNA, CSc.**
ŠKOLITEL

BRNO 2020

Abstract

The present thesis introduces and studies new possibilities of parallel processing in formal models. More specifically, it focuses its attention on parallel versions of jumping finite automata and on normal forms of grammars with interesting parallel properties.

In the first part of this thesis, we give an initial motivation for studying parallel processing in formal models. We briefly introduce jumping models and normal forms of grammars and grammar systems. Finally, we state the precise focus and goals of our research.

The second part of this thesis is focused on new results on jumping finite automata. First, we introduce n-parallel jumping finite automata that enhance the original jumping finite automaton model with multiple reading heads. The rest of the chapter then studies the accepting power of the model under two different jumping modes. Second, we introduce double-jumping finite automata and explore advanced jumping modes utilizing two heads. We study the accepting power of the models and also the closure properties of the related language families. Lastly, we introduce jumping $5' \to 3'$ Watson-Crick finite automata that combine the jumping behavior with the biology-inspired Watson-Crick finite automata that process double-stranded DNA sequences. The rest of this chapter then studies the accepting power of the model under unrestricted and various restricted conditions.

The third part of this thesis is focused on new results on CD grammar systems. We introduce two types of transformations that turn arbitrary general grammars into equivalent two-component general CD grammar systems of very reduced and simplified forms. Apart from the reduction and simplification, we describe several other useful properties concerning these systems and the way they work.

In the last part, we mention possible application perspectives for the introduced models and normal forms, and we conclude the thesis with the final summary and the description of theoretical perspectives for the achieved results.

Keywords

parallel processing, discontinuous tape reading, parallel tape reading, normal forms, simulated non-context-free rules, homogeneous rules, evenly homogeneous rules, general grammars, jumping finite automata, left and right jumps, n-parallel right linear grammars, even-length languages, Watson-Crick finite automata, CD grammar systems

Reference

KOCMAN, Radim. *On Parallel Processing in Formal Models: Jumping Automata and Normal Forms*. Brno, 2020. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. RNDr. Alexander Meduna, CSc.

Abstrakt

Tato disertační práce představuje a zkoumá nové možnosti paralelního zpracování ve formálních modelech. Zaměřuje se přitom především na paralelní verze skákajících konečných automatů a na normální formy gramatik se zajímavými paralelními vlastnostmi.

První část práce popisuje motivaci pro studium paralelního zpracování ve formálních modelech a stručně představuje skákající modely a normální formy gramatik a gramatických systémů. Jsou zde také upřesněny cíle prezentovaného výzkumu.

Druhá část práce je zaměřena na prezentaci nových výsledků v oblasti skákajících konečných automatů. Jako první je zde přestaven n-paralelní skákající konečný automat, který rozšiřuje původní model skákajícího konečného automatu a podporu většího množství čtecích hlav. Práce následně studuje sílu tohoto modelu ve dvou rozdílných skákajících módech. Následuje představení dvojitě skákajících konečných automatů, u kterých jsou zkoumány pokročilé skákající módy využívající dvě čtecí hlavy. Kromě síly těchto modelů jsou zde zkoumány i uzávěrové vlastnosti příslušných tříd jazyků. Jako poslední jsou v této části představeny skákající $5' \to 3'$ Watson-Crick konečné automaty, které kombinují skákající chování s biologií inspirovanými Watson-Crick konečnými automaty. Opět je zde zkoumána síla tohoto modelu a to i s uvážením rozličných omezujících podmínek.

Třetí část práce je zaměřena na prezentaci nových výsledků v oblasti CD gramatických systémů. Jsou zde prezentovaný dva typy transformací, které dokáží převést libovolnou obecnou gramatiku na dvoukomponentový obecný CD gramatický systém velmi redukované a zjednodušené formy. Kromě této významné redukce a zjednodušení prezentuje práce i několik dalších užitečných vlastností souvisejících s těmito systémy.

V poslední části textu jsou pak nastíněny možné oblasti využití představených modelů a normálních forem. Práce je následně uzavřena souhrnem dosažených výsledků a závěrečnými poznámkami k dalšímu směřování výzkumu.

Klíčová slova

paralelní zpracování, nespojité čtení pásky, paralelní čtení pásky, normální formy, simulace kontextových pravidel, homogenní pravidla, souměrná homogenní pravidla, obecné gramatiky, skákající konečné automaty, levé a pravé skoky, n-paralelní pravě lineární gramatiky, jazyky s řetězci sudé délky, Watson-Crick konečné automaty, CD gramatické systémy

Citace

KOCMAN, Radim. *On Parallel Processing in Formal Models: Jumping Automata and Normal Forms*. Brno, 2020. Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Školitel prof. RNDr. Alexander Meduna, CSc.

On Parallel Processing in Formal Models: Jumping Automata and Normal Forms

Declaration

I hereby declare that this thesis is my own work that has been created under the supervision of Alexander Meduna. It is largely based on the following seven publications where I am the main contributor, which I wrote jointly with Zbyněk Křivka, Alexander Meduna, and Benedek Nagy:

(1) Kocman, R. n-Parallel Jumping Finite Automata. In: *Excel@FIT 2015*. 2015. ([32]),

(2) Kocman, R. and Meduna, A. On Parallel Versions of Jumping Finite Automata. In: *Proceedings of the 2015 Federated Conference on Software Development and Object Technologies, SDOT 2015*. Springer International Publishing, 2016, p. 142–149. Advances in Intelligent Systems and Computing, vol. 511. ([37]),

(3) Kocman, R., Křivka, Z. and Meduna, A. On Double-Jumping Finite Automata. In: *Eighth Workshop on Non-Classical Models of Automata and Applications, NCMA 2016*. Osterreichische Computer Gesellschaft, 2016, p. 195–210. books@ocg.at, vol. 321. ([33]),

(4) Kocman, R., Nagy, B., Křivka, Z. and Meduna, A. A Jumping $5' \to 3'$ Watson-Crick Finite Automata Model. In: *Tenth Workshop on Non-Classical Models of Automata and Applications, NCMA 2018*. Osterreichische Computer Gesellschaft, 2018, p. 117–132. books@ocg.at, vol. 332. ([38]),

(5) Kocman, R., Křivka, Z. and Meduna, A. On Double-Jumping Finite Automata and Their Closure Properties. *RAIRO-Theor. Inf. Appl.* 2018, vol. 52, 2-3-4, p. 185–199. ([34]),

(6) Kocman, R., Křivka, Z. and Meduna, A. General CD Grammar Systems and Their Simplification. *Journal of Automata, Languages and Combinatorics*. 2020, vol. 25, no. 1, p. 37–54. ([35]),

(7) Kocman, R., Křivka, Z., Meduna, A. and Nagy, B. A Jumping $5' \to 3'$ Watson-Crick Finite Automata Model. *Acta Informatica*. (in review). ([36]).

I have listed all the literary sources, publications, and other sources which were used during the preparation of this thesis.

.....
Radim Kocman
September 2, 2020

Acknowledgements

I would especially like to thank my coauthors Zbyněk Křivka, Alexander Meduna, and Benedek Nagy for their invaluable help during the whole research. I would also like to thank all anonymous referees for their constructive suggestions and comments that helped improve the content of our papers. And last but not least, I am grateful to my family for the long years of support and care.

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070); The Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science – LQ1602; the TAČR grant TE01020415; the BUT grant FIT-S-14-2299; and the BUT grant FIT-S-17-3964.

From a more general perspective, I would like to acknowledge the work of Brady Haran (www.bradyharan.com); especially his enthusiasm to popularize even the theoretical branches of mathematics and computer science. His videos and podcasts with famous scientist are a great source of motivation for further research in these theoretical fields.

Finally, to lighten up the heavy mathematical content of this thesis, I would like to mention the work of the surreal comedy group Monty Python. Even though the Pythons were not scientists themselves, they pushed the boundaries of what is possible in television comedy. Even today, their work still represents the pinnacle of comedy that many people try to reach. To acknowledge the recent 50 years of Monty Python, let us celebrate with some famous quotes that are mildly relevant to the content of the following chapters.

Contents

Part I

Introduction and Terminology

It's...
—Monty Python's Flying Circus

Chapter 1

Introduction

In this chapter, we give an initial motivation for studying parallel processing in formal models. We briefly introduce jumping models—quite a new group of formal models focused on discontinuous information processing, which were not yet studied together with parallel mechanisms—and normal forms of grammars and grammar systems—the common unifying forms of definitions of formal models, which are usually not concerned with parallelism. Afterwards, we state our focus and goals for studying parallelism together with jumping automata and normal forms. Finally, we outline the organization of the rest of this work. This thesis assumes that the reader is firmly familiar with the basic notions from the theory of automata and formal languages, and thus we use them here extensively without further explanation. The more advanced terminology that will be used later in the results of this work is introduced in greater detail in Chapter 2.

1.1 Parallelism

When we talk about parallelism in modern computer science, we almost automatically mean some form of parallel processing or parallel computing. By these terms we refer to situations where we want to split some large task into smaller chunks of work in such a way that the chunks can be executed in parallel on separate processing units, and the whole task can thus be computed faster than if it was executed completely sequentially on a single processing unit.

Nonetheless, this perception of the notion of parallelism can change quite rapidly when we wander into more theoretical branches of computer science; especially if we consider the basic research in the theory of formal languages. There are many formal models in this area that incorporate some form of parallelism, but they utilize very diverse mechanics in the background to achieve their goal. If we take a broader look at these formal models and the basic research in general, we can roughly divide parallelism in formal language theory into the following three categories:

(P.1) parallelism that increases the expressive power of the model,

(P.2) parallelism that is a fundamental part of the behavior of the model,

(P.3) parallelism that splits the work of the task.

Parallelism That Increases the Expressive Power of the Model

The most commonly studied category in the basic research is probably category (P.1). This is especially noticeable in formal grammars. Consider classical formal grammars in general, there is a big difference if a model can use only context-free rules or also non-context-free rules. It is much harder to deal with the non-context-free rules from both the theoretical and practical point of view. Therefore, there is a large incentive to study models that can use only the context-free (or even more restricted) rules but that also incorporate some additional mechanisms which further increase their generative power.

In formal grammars, the models can incorporate parallelism in such a way that, in each step of the rewriting process, the grammar rewrites several symbols in the sentential form at once in parallel. Let us mention some well-known models that match this description:

- scattered context grammars (see [21, 58, 54]),
- simple matrix grammars (see [25, 74, 89, 90]),
- equal matrix grammars (see [80]),
- n-parallel (right-)linear grammars (see [75, 89, 90, 88, 73, 74]).

In the case of finite automata, we can imagine the parallelism of category (P.1) as a parallel cooperation of multiple heads. There are several well-known models of finite automata that utilize more than one head, nonetheless, their behavior do not fall precisely into one specific category of parallelism; so we will leave their description for later.

A very common property of models from this category is that we can freely select their degree of parallelism. More specifically, we can choose n which represents the number of symbols or heads that are considered together in a single step of the model. Then, if $n = 1$, we get the power of a classical non-augmented model (e.g., the power of context-free grammars); and, for $n > 1$, we either get an infinite hierarchy of more powerful models or the power of the model increases at first but then stabilizes. Due to this common property, we can also include parallel communicating (PC) grammar systems (see [8, 78]) into this category since they behave very similarly in this regard.

Parallelism That Is a Fundamental Part of the Behavior of the Model

Considering category (P.2), we are looking at the models that have parallelism rooted inseparably into their core structure. From our exploration of this topic, it seems that the models which fall into this category are usually related to biology.

On the one hand, there are massively parallel models such as Lindenmayer systems (see [77, 58, 54]) that are based on the evolution process. In these models, all eligible symbols in the sentential form are always rewritten together at once in parallel. Consequently, it is not possible to select a constant degree of parallelism for these models since the conditions continuously change depending on the current task.

On the other hand, there are also models with a fixed degree of parallelism such as Watson-Crick finite automata (see [72]). These automaton models use two heads in parallel in such a way that each head processes one strand of a double-stranded DNA input sequence. Consequently, the degree of parallelism of Watson-Crick finite automata is always two.

Parallelism That Splits the Work of the Task

Lastly, if we consider category (P.3) in the basic research, it seems that there is not much interest to study possibilities how to split the work for the given tasks. This may not be that surprising because in the basic research we usually study characteristics like the expressive power, closure properties, and the decidability and complexity of various operations; and, of course, these results are not affected by parallelism. We often even prefer approaches that are completely sequential because it makes the subsequent proof techniques much easier in many cases. When we do consider parallelism that splits the work of the tasks (see [77, 78]), we usually just simply conclude that if the model behaves nondeterministically, then we can explore different cases in parallel, and if the model uses only context-free rules, then we can trivially split the generation process into multiple independent parts.

It is possible to find some theoretical papers that explore this role of parallelism further in certain areas, e.g., in biomolecular computing (see [43]); but a thorough study is usually left for practical applications such as parsing (see [22]), formal verification, and others.

Parallelism and Finite Automata

The situation around the types of parallelism gets more complex if we look at finite automata. Thus, we introduce some additional categorization.

There are some finite automaton models that have the same expressive power as grammars from category (P.1). For example, self-regulating finite automata (see [52]), pure multi-pushdown automata that perform complete pushdown pops (see [48]), and finite-turn checking automata (see [81]), which are connected to the various versions of simple matrix, equal matrix, and n-parallel right-linear grammars. However, we do not consider these models to be parallel. This is due to the fact that, up until quite recently, automaton models always read the input tape almost exclusively in the strictly continuous (left-to-right) symbol-by-symbol way. The mentioned models are no exceptions, and thus they use various kinds of stacks to match the expressive power of the parallel grammars but otherwise work strictly continuously on the input tape in a completely non-parallel way.

As we have already pointed out, we can imagine parallelism in finite automata as a parallel cooperation of multiple heads. There is indeed the well-known concept of Turing machines with multiple tapes and multiple heads; which was also adapted and studied in terms of finite automaton models. Nonetheless, not all such models necessarily work in a parallel way. Considering multi-head finite automata that actually do work in a parallel way, we can find two distinct categories of their behavior:

(PA.1) multi-head automata where each head works on an independent copy of the input,

(PA.2) multi-head automata where heads cooperate to process the single input.

The first category seems to be the most studied one so far. Let us mention some prominent models that fit into this description: classical Watson-Crick finite automata (see [72]), multi-head finite automata (see [76, 28, 83, 24]), and parallel communicating finite automaton systems (see [24]). In these models, the heads can work in parallel, however, their behavior can be hardly seen as parallel processing since it does not speed up the task in any way. In most cases, there is a single read-only input tape that must be completely traversed with all heads until the conclusion about the acceptance of the input is reached.

We only know about a few models that fall into the second category. These are finite automaton models introduced by Nagy that utilize two heads with the following behavior.

The first head reads the input from left to right, the second head reads the input from right to left, and the processing of the input ends when the heads meet each other on the tape. This concept was explored several times in various models:

- 2-head finite automata (see [64]),
- $5' \to 3'$ Watson-Crick finite automata (see [60, 61, 62, 63, 65, 70, 71, 69]),
- multicounter $5' \to 3'$ Watson-Crick finite automata (see [11, 59, 23]),
- two-head finite-state acceptors with translucent letters (see [67, 68]).

In these models, the heads truly cooperate in parallel on a single tape; thus, this behavior can be seen as parallel processing. Naturally, their degree of parallelism is always two.

1.2 Jumping Models

The idea of a jumping mechanism that is integrated deeply into the core behavior of formal models is quite a new concept that was first proposed in 2012 by Meduna and Zemek in [57]. The main motivation behind this concept is the fact that in the previous century most classical computer science methods were developed for continuous information processing, but in modern computation methods we often process information in a discontinuous way. The continuous processing approach is deeply rooted in classical formal models such as finite automata which traditionally process the input information in a strictly continuous left-to-right symbol-by-symbol way. Therefore, it makes sense to introduce and study jumping mechanisms that can more appropriately represent the behavior of modern computation methods that often have to jump over large portions of the input information between individual steps of the process.

In the following years, this idea got a lot of traction among other researchers in the field of formal language theory. At the time of writing, there are around 30 papers that study jumping models in various ways, and this number is still increasing. From the theoretical point of view, these models have an interesting characteristic that, on the one hand, they often define language families that are outside the usual Chomsky hierarchy, but, on the other hand, they are still related to some other well-known mathematical models. With this characteristic, it is possible to combine results from different fields that previously looked unrelated. It is out of the scope of this thesis to cover all studied models, but we at least give a brief overview of the most influential ones.

Jumping Finite Automata

The definition of a jumping finite automaton was first introduced by Meduna and Zemek in [57], and it can be also found in the follow-up books [58, 54].

Let us first recall the notion of a classical finite automaton, M, which consists of an input tape, a reading head, and a finite state control. The input tape is divided into squares. Each square contains one symbol of an input string. The symbol under the reading head, a, is the current input symbol. The finite control is represented by a finite set of states together with a control relation, which is usually specified as a set of computational rules. The automaton M computes by making a sequence of moves. Each move is made according to a computational rule that describes how the current state is changed and whether the current input symbol is read. If the symbol is read, the reading head is shifted precisely one

square to the right. M has one state defined as the start state and some states designated as final states. If M can read w by making a sequence of moves from the start state to a final state, M accepts w; otherwise, M rejects w.

In essence, a jumping finite automaton works just like a classical finite automaton except it does not read the input tape in a symbol-by-symbol left-to-right way. After the automaton reads a symbol, the head can jump over (skip) a portion of the tape in either direction. Once an occurrence of a symbol is read on the tape, it cannot be re-read again later. Otherwise, it coincides with the standard notion of a finite automaton.

Apart from the definition, the paper [57] studies the accepting power, decidability properties, and closure properties of the model under various restrictions. Surprisingly, compared to classical finite automata, there is a significant difference if the model is a general jumping finite automaton (GJFA), which can read multiple symbols in a step, or a non-general jumping finite automaton (JFA), which can read only a single symbol in a step.

Concerning GJFAs, there are papers written by Vorel (see [84, 85, 86]) that continue the investigation of decidability and closure properties. Moreover, they connect GJFAs with graph-controlled insertion systems and Galiukschov semicontextual grammars.

Concerning both GJFAs and JFAs, there are papers written by Fernau, Paramasivan, Schmid, and Vorel (see [14, 15]) that present a large number of various new results and also connect JFAs with shuffle languages, commutative context-free grammars, letter bounded languages, and regular expressions over comutative monoids.

Lastly, concerning JFAs, there are papers written by Beier, Holzer, and Kutrib (see [4, 5]) that study their operational state complexity and decidability and also connect JFAs with semilinear sets and Parikh images of regular sets.

Jumping Grammars

Jumping grammars can be seen as a transformation of jumping finite automata into the form of formal grammars. Their definition was first introduced in 2015 by Křivka and Meduna in [31], and it can be also found in the follow-up book [54].

Let us first recall the notion of a classical grammar, G, which represents a language-generating rewriting system based upon an alphabet of symbols and a finite set of productions. The alphabet of symbols is divided into two disjoint sub-alphabets of terminal and nonterminal symbols. Each production rule represents a pair of the form (x, y), where x and y are strings over the alphabet of G; we can write (x, y) as $x \to y$. Starting from a special start nonterminal symbol, G repeatedly rewrites strings according to its production rules until it obtains a sentence—that is, a string that solely consists of terminal symbols. The set of all sentences represents the language generated by the grammar. In greater detail, G rewrites a string z according to $x \to y$ so it (1) selects an occurrence of x in z, (2) erases it, and (3) inserts y precisely at the position of this erasure. More formally, let $z = uxv$, where u and v are strings. By using $x \to y$, G rewrites uxv to uyv.

The notion of a jumping grammar is conceptualized just like that of a classical grammar; however, it rewrites strings in a slightly different way. Consider G, described above, as a jumping grammar. Let z and $x \to y$ have the same meaning as above. The jumping grammar G rewrites a string z according to $x \to y$ so it performs (1) and (2) as described above, but, during (3), G can jump over a portion of the rewritten string in either direction and insert y there. More formally, by using $x \to y$, G rewrites ucv as udv, where u, v, w, c, d are strings such that either (i) $c = xw$ and $d = wy$ or (2) $c = wx$ and $d = yw$. Otherwise, it coincides with the standard notion of a grammar.

Apart from the definition, paper [31] mainly studies the generative power of a large number of various types of jumping grammars. Moreover, the paper shows that jumping finite automata and jumping grammars have connection to multisets (see [82, 39, 40, 10]). Additionally, there are two papers written by Madejski (see [44, 45]) that introduce jumping and pumping lemmas and connect jumping grammars with permutation grammars.

One-way Jumping Finite Automata

From the definitions of the previous two models, it may seem that jumping models have to be inherently nondeterministic. However, this is not the case as it is shown by one-way jumping finite automata introduced in 2015 by Chigahara, Fazekas, and Yamamura in [6, 7].

One-way jumping finite automata make moves similar to jumping finite automata but with some changes leading to a deterministic behavior. The reading head moves only in one direction and starts at the beginning of the input tape. It moves from left to right (and possibly jumps over parts of the input), and, upon reaching the end of the input tape, it is returned to the beginning of the input; continuing the computation until all the symbols are read or the automaton gets stuck in a state in which it cannot read any symbol of the remaining input. If a transition is defined for the current state and the next symbol to be read, then the automaton reads the symbol. If not, but in the remaining input there are symbols for which a transition is defined from the current state, the reading head jumps to the nearest such a symbol to the right.

Apart from the definition, paper [7] studies the accepting power and closure properties, and it also defines pumping lemmas for the resulting language families. In [13], Fazekas and Yamamura study sufficient conditions for the resulting language to be regular. There are also papers written by Beier and Holzer (see [2, 1, 3]) that study inclusion relations, closure properties, and decidability properties. Lastly, paper [12] written by Fazekas, Hoshi, and Yamamura compares the deterministic and nondeterministic finite automata and pushdown automata when they use standard, jumping, and one-way jumping steps.

Other Jumping Models

Besides the most influential models mentioned previously, there are also other papers that study the jumping mechanism further in more advanced formal models:

- two-dimensional jumping finite automata (see [26, 46, 27]),
- jumping scattered context grammars (see [53, 54]),
- jumping pure grammars (see [29]),
- jumping restarting automata (see [87]),
- jumping multi-head automata (see [41]),
- Watson-Crick jumping finite automata (see [47]).

Note that it may seem, from the names of jumping multi-head automata and Watson-Crick jumping finite automata, that these models are similar to the models studied later in this thesis. However, both of the mentioned models fall into the category (PA.1) of parallelism in finite automata. On the other hand, all finite automata studied in this thesis fall into the category (PA.2) which is a fundamentally different behavior.

1.3 Normal Forms and Grammar Systems

Moving away from the idea of the jumping mechanism, we need to introduce the remaining two concepts that are also studied together with parallelism in this thesis.

Normal Forms

As we have shown previously, a classical (general) grammar G contains production rules of the form $x \to y$, where x and y are strings over the alphabet of G. If we want to change the generative power of the grammar, we can put restrictions on the form of the rules. Classically, we consider some types of monotonous, context-sensitive, context-free, ε-free, linear, right-linear, and regular restrictions (see, e.g., [77, 31]). Nonetheless, even in these cases, the forms of rules are often still rather loose. This can be an unwanted property from both the theoretical and practical point of view because the follow-up proofs and algorithms have to take into account all possible forms of the definition of the grammar. Therefore, there is an incentive in formal language theory to study normal forms of grammars and grammar systems that severely restrict the possible forms of the definition of the model but, in the same time, keep the generative power intact.

We skip the description of basic normal forms that handle only grammars with context-free rules since, in these cases, it is rather easy to work with them in a parallel way. However, let us take a look at some well-known normal forms for general grammars (see, e.g., [58]). In all presented normal forms, G is a grammar, S, A, B, C, D denote nonterminals, a denotes a terminal, and ε denotes an empty string.

- *Kuroda normal form*—G is in Kuroda normal form if every rule has one of the forms: (1) $AB \to CD$, (2) $A \to BC$, (3) $A \to a$, or (4) $A \to \varepsilon$.
- *Penttonen normal form*—G is in Penttonen normal form if every rule has one of the forms: (1) $AB \to AC$, (2) $A \to BC$, (3) $A \to a$, or (4) $A \to \varepsilon$.
- *Geffert normal forms*—G is in one of Geffert normal forms if S is the start nonterminal symbol, the context-free rules are of the form (1) $S \to uSa$, (2) $S \to uSv$, (3) $S \to uv$, and one of the following holds:

  (G.1) G contains the nonterminals S, A, B, C,
  G contains the non-context-free rule $ABC \to \varepsilon$,
  $u \in \{A, AB\}^*$ and $v \in \{BC, C\}^*$,

  (G.2) G contains the nonterminals S, A, B, C, D,
  G contains the non-context-free rules $AB \to \varepsilon$ and $CD \to \varepsilon$,
  $u \in \{A, C\}^*$ and $v \in \{B, D\}^*$,

  (G.3) G contains the nonterminals S, A, B,
  G contains the non-context-free rule $ABBBA \to \varepsilon$,
  $u \in \{AB, ABB\}^*$ and $v \in \{BBA, BA\}^*$.

These normal forms are all frequently used in formal language theory. However, we argue that none of them has particularly fitting parallel properties. When we want to construct a parallel rewriting process for general grammars, the non-context-free rules really complicate the task since there is no simple way how to split the generation of a sentence into multiple independent parts, and the classical normal forms do not help with this matter.

First, consider an unrestricted general grammar. There can be a large number of non-context-free rules. These rules can work with very large contexts since there is no bound on the length of x in $x \to y$. Furthermore, the non-context-free rules can be also used anywhere in the generation process.

Second, consider Kuroda and Penttonen normal forms. Indeed, the required context of the non-context-free rules is now minimal. However, there can still be a large number of these rules, and they can still be used anywhere in the generation process.

Lastly, consider Geffert normal forms. There is a limited number of non-context-free rules, and they work with small contexts. Nonetheless, all Geffert normal forms share the same deeply-rooted property that, in any generated string, there is always at most one position that can be rewritten with the rules of the grammar. In some situations, this property can be highly valuable from both the theoretical and practical point of view; however, this complicates the construction of a parallel rewriting process even further.

Besides the classical normal forms, we can find many other normal forms for various formal models (see, e.g., [58, 54, 51]). However, it seems that in almost all cases the definitions are primarily focused only on the very restricted forms of rules, minimum number of non-context-free rules, and minimum number of nonterminals. Consequently, they do not care about the resulting parallel properties.

CD Grammar Systems

In essence, a cooperating distributed (CD) grammar system (see [8]) can be seen as an extension of a classical grammar. It has not one but multiple finite sets of production rules (components), and the rewriting process can operate in various complex modes that control which sets of production rules can be currently used. From another perspective, a CD grammar system can be seen as a group of grammars that distribute their work and cooperate on a single string to produce the final sentence.

Considering the core behavior of CD grammar systems, their extension over general grammars is not parallel in nature because the modes switch between components in a strictly sequential way. However, we find this model useful for the study of parallel properties of normal forms. To give a brief insight, with CD grammar systems, we can strictly split the context-free and non-context-free rules into different components, and we can clearly divide the rewriting process into several phases that use different types of rules. This can help us to pinpoint opportunities for a viable parallel rewriting process.

1.4 Specification of Goals

The principal focus of this thesis is the theoretical study of parallelism in the areas of formal language theory where this approach was not yet thoroughly considered. First, we explore parallel processing with jumping finite automata. Second, we introduce new normal forms designed for parallel rewriting.

New Results on Jumping Automata

The first unexplored area can be easily seen if we take a closer look at the previous description of parallelism in classical finite automata and the new jumping mechanism introduced in jumping finite automata. Once we shift our attention to discontinuous information processing, and we are no longer restricted with the classical reading in a continuous left-

to-right symbol-by-symbol way, there are a lot of new opportunities how to design the behavior of multi-head finite automaton models. From the point of view of parallelism in finite automata, we want to focus on category (PA.2) where the heads work in parallel to process the single input. From the point of view of general parallelism, we want to design models that fit into category (P.3) but also share some similarities with categories (P.1) and (P.2). To be more precise, we will introduce and study new parallel versions of jumping finite automata. Since these mechanisms were not yet studied together, this research should lead to some novel results that are usually not observed in classical models. Moreover, we should be able to find some new close connections with different formal models.

New Results on CD Grammar Systems

The second unexplored area was already foreshadowed in the description of normal forms of grammars and grammar systems. We want to introduce new normal forms that are focused not just on the usual restrictive properties but also on the resulting parallel properties. More precisely, we will use an extended version of CD grammar systems that can accept recursively-enumerable languages, and we will introduce new normal forms for these grammar systems that will have a very limited number of non-context-free rules and that will be suitable for a parallel rewriting process. Such normal forms can be interesting from both the theoretical and practical point of view. This thesis is focused primarily on the theoretical aspects of the topic, but we will also mention some further ideas in the conclusion.

1.5 Organization

The content of this thesis is divided into four parts and eight chapters. Part I serves mainly as an introduction to the topic. After the current Chapter 1, Chapter 2 presents all the terminology used in the following results.

Part II covers all results regarding jumping finite automata. Chapter 3 studies the initial general version of parallel jumping finite automata—n-parallel jumping finite automata. Chapter 4 continues the study with the follow-up version of parallel jumping finite automata that is focused on advanced reading modes with two heads—double-jumping finite automata. Chapter 5 explores a combined model of jumping finite automata and Watson-Crick finite automata—jumping $5' \to 3'$ Watson-Crick finite automata.

Part III covers results regarding CD grammar systems. Chapter 6 explores normal forms of general CD grammar systems with useful parallel properties.

Part IV closes the thesis. Chapter 7 explores the application perspectives of the introduced models and normal forms. Chapter 8 presents the final summary and theoretical perspectives of the achieved results.

We use only the finest baby frogs...
—Monty Python's Flying Circus

Chapter 2

Terminology

This thesis assumes that the reader is familiar with the theory of automata and formal languages (see [49, 91]). This chapter recalls only the crucial notions used in the presented results. Nonetheless, the content of this thesis covers a large number of different models with various backgrounds. Therefore, even if we recall only the crucial notions, the preliminaries are rather lengthy. The reader is advised to first take only a quick look at the overall terminology and then return for more details later when the models are actually used. At the beginnings of the following chapters we always reference the necessary terminology.

2.1 General Notions and Operations

This section describes the general terminology around sets, strings, and their basic operations, which is used in all results throughout the thesis.

2.1.1 Sets and Strings

For a set Q, $\mathrm{card}(Q)$ denotes the cardinality of Q, and 2^Q denotes the power set of Q. For an alphabet (finite nonempty set) V, V^* represents the free monoid generated by V under the operation of concatenation. The unit of V^* is denoted by ε. Members of V^* are called *strings*. Set $V^+ = V^* - \{\varepsilon\}$; algebraically, V^+ is thus the free semigroup generated by V under the operation of concatenation. For $x \in V^*$, $|x|$ denotes the length of x, $\mathrm{rev}(x)$ denotes the reversal of x, and $\mathrm{alph}(x)$ denotes the set of all symbols occurring in x; for instance, $\mathrm{alph}(0010) = \{0, 1\}$. For $x \in V^*$ and $a \in V$, $|x|_a$ denotes the number of occurrences of a in x. Let X and Y be sets; we call X and Y to be *incomparable* if $X \not\subseteq Y$, $Y \not\subseteq X$, and $X \cap Y \neq \emptyset$.

2.1.2 Mirror Image

Let $x = a_1 a_2 \cdots a_n$, where $a_i \in V$, $1 \leq i \leq n$, $n \geq 0$ ($x = \varepsilon$ if and only if $n = 0$). The *mirror image* of x, denoted by $\mathrm{mi}(x)$, is defined as $\mathrm{mi}(x) = a_n a_{n-1} \cdots a_1$ ($\mathrm{mi}(x) = \mathrm{rev}(x)$). For $L \subseteq V^*$, we define $\mathrm{mi}(L) = \{\mathrm{mi}(x) : x \in L\}$.

2.1.3 Parikh Vector

The *Parikh vector* associated to a string $x \in V^*$ with respect to the alphabet $V = \{a_1, a_2, \ldots, a_n\}$ is $\Psi_V(x) = (|x|_{a_1}, |x|_{a_2}, \ldots, |x|_{a_n})$. For $L \subseteq V^*$ we define $\Psi_V(L) = \{\Psi_V(x) : x \in L\}$.

2.1.4 Shuffle

For $x, y \in V^*$, the *shuffle* of x and y, denoted by $\text{shuffle}(x, y)$, is defined as $\text{shuffle}(x, y) = \{x_1y_1x_2y_2 \cdots x_ny_n : x = x_1x_2 \cdots x_n,\ y = y_1y_2 \cdots y_n,\ x_i, y_i \in V^*,\ 1 \leq i \leq n,\ n \geq 1\}$. For $L_1, L_2 \subseteq V^*$, $\text{shuffle}(L_1, L_2) = \{z : z \in \text{shuffle}(x, y),\ x \in L_1,\ y \in L_2\}$.

2.2 Grammars and Languages

This section describes the basic terminology around grammars and languages. Furthermore, it establishes notation for various language families from Chomsky hierarchy and their derived subfamilies which are heavily referenced in the whole thesis. Lastly, we present precise formal definitions for several normal forms and restrictions of grammars that are vital for the presented results.

2.2.1 Basic Definitions

A *general grammar* or, more simply, a *grammar* is quadruple $G = (N, T, P, S)$, whose components are defined as follows. N and T are alphabets such that $N \cap T = \emptyset$. Symbols in N are referred to as *nonterminals*, while symbols in T are referred to as *terminals*. $S \in N$ is the start symbol of G. P is a finite set of *(general) rules* of the form $x \to y$, where $x, y \in (N \cup T)^*$ and $\text{alph}(x) \cap N \neq \emptyset$. For brevity, we sometimes denote a rule $x \to y$ with a unique label p as $p : x \to y$, and, instead of $p : x \to y \in P$, we simply write $p \in P$. The left-hand side x and the right-hand side y of p are denoted by $\text{lhs}(p)$ and $\text{rhs}(p)$, respectively. If $p \in P$ and $|\text{rhs}(p)| = 0$, it is an ε-rule. The rule $p \in P$ is considered context-free if $|\text{lhs}(p)| = 1$; otherwise, it is a non-context-free rule. Set $\text{ContextFree}(P) = \{p \in P : |\text{lhs}(p)| = 1\}$ and $\text{NonContextFree}(P) = \{p \in P : |\text{lhs}(p)| \geq 2\}$. If $x \to y \in P$ and $u, v \in (N \cup T)^*$, then $uxv \Rightarrow uyv\ [x \to y]$, or simply $uxv \Rightarrow uyv$. In the standard manner, let us extend $\Rightarrow$ to $\Rightarrow^n$, where $n \geq 0$; then, based on $\Rightarrow^n$, let us define $\Rightarrow^+$ and $\Rightarrow^*$. The language generated by G, $L(G)$, is defined as $L(G) = \{w \in T^* : S \Rightarrow^* w\}$.

We recognize several special cases of grammars:

- G is a *context-sensitive grammar* if every $x \to y \in P$ satisfies either $x = \alpha A \beta$ and $y = \alpha v \beta$ such that $A \in N$, $\alpha, \beta, v \in ((N - \{S\}) \cup T)^*$, $v \neq \varepsilon$; or $x = S$ and $y = \varepsilon$.
- G is a *context-free grammar* if every $x \to y \in P$ satisfies $x \in N$.
- G is a *linear grammar* if every $x \to y \in P$ satisfies $x \in N$ and $y \in T^*NT^* \cup T^*$.
- G is a *right-linear grammar* if every $x \to y \in P$ satisfies $x \in N$ and $y \in T^*N \cup T^*$.
- G is a *regular grammar* if every $x \to y \in P$ satisfies $x \in N$ and $y \in TN \cup T \cup \{\varepsilon\}$.

A language L is recursively enumerable, context-sensitive, context-free, linear, or regular (right-linear) if and only if $L = L(G)$, where G is a general, context-sensitive, context-free, linear, or regular (right-linear) grammar, respectively.

Let **RE**, **CS**, **CF**, **LIN**, **REG**, and **FIN** denote the families of recursively enumerable, context-sensitive, context-free, linear, regular, and finite languages, respectively.

2.2.2 Derived Language Families

Sometimes it is useful for our results to consider additional derived language families that do not precisely match the Chomsky hierarchy. Let $\mathbf{CS}_{\text{even}}$, $\mathbf{CF}_{\text{even}}$, $\mathbf{LIN}_{\text{even}}$, $\mathbf{REG}_{\text{even}}$,

and $\mathbf{FIN}_{\text{even}}$ denote the appropriate subfamilies of the traditional language families that contain only languages with even-length strings. Let $\mathbf{FIN}_{\varepsilon\text{-inc}}$ denote the family of finite languages that always contain the empty string.

2.2.3 Endmarking Closure

Let $\mathscr{L}$ be a language family. We say that $\mathscr{L}$ is *closed under endmarking* if and only if for every $L \in \mathscr{L}$, where $L \subseteq V^*$, for some alphabet V, $\# \notin V$ implies that $L\{\#\} \in \mathscr{L}$. We also say that $\mathscr{L}$ is *closed under endmarking on both sides* if and only if the previous implies that $\{\#\}L\{\#\} \in \mathscr{L}$.

2.2.4 Kuroda Normal Form

Let $G = (N, T, P, S)$ be a grammar. G is in *Kuroda normal form* (see Section 8.3.3. in [49]) if every rule $p \in P$ has one of these three forms: (1) $AB \to CD$, (2) $A \to BC$, or (3) $A \to a$, where $A, B, C, D \in N$ and $a \in (T \cup \{\varepsilon\})$.

2.2.5 Homogeneous Restrictions

Let $G = (N, T, P, S)$ be a grammar. If $x \to y \in P$ and $x \in \{A\}^+$ for some $A \in N$, then $x \to y$ is a *homogeneous rule* (see [51]). Furthermore, if also $y \in \{B\}^+$ for some $B \in (N \cup T)$ and $|x| = |y|$, then $x \to y$ is an *evenly homogeneous rule*. G is a *homogeneous grammar* if every $p \in P$ is homogeneous.

2.3 n-Parallel Right-Linear Grammars

For $n \geq 1$, an *n-parallel right-linear grammar* (see [75, 90, 88, 73, 74]), an *n-PRLG* for short, is an $(n+3)$-tuple $G = (N_1, \ldots, N_n, T, S, P)$, where N_i, $1 \leq i \leq n$, are mutually disjoint nonterminal alphabets (and we denote $\bigcup_{i=1}^{n} N_i$ by N), T is a terminal alphabet, $T \cap N = \emptyset$, $S \notin (T \cup N)$ is the sentence symbol, and P is a finite set of pairs. Members of P are referred to as rules of G, and, instead of $(X, x) \in P$, we write $X \to x \in P$. Each rule in P has one of the following forms:

$$\begin{aligned}
&(1)\ S \to X_1 \cdots X_n, && X_i \in N_i,\ 1 \leq i \leq n,\\
&(2)\ X \to aY, && X, Y \in N_i,\ \text{for some } i,\ 1 \leq i \leq n,\ a \in T^*,\\
&(3)\ X \to a, && X \in N,\ a \in T^*.
\end{aligned}$$

The binary yield relation, symbolically denoted by $\Rightarrow$, is defined as follows. Let $x, y \in (N \cup \{S\} \cup T)^*$ then $x \Rightarrow y$ if and only if

$$\begin{aligned}
&\text{either } x = S \text{ and } S \to y \in P\\
&\quad \text{or } x = a_1 X_1 \cdots a_n X_n,\ y = a_1 x_1 \cdots a_n x_n,\\
&\qquad \text{where } a_i \in T^*,\ X_i \in N_i,\ \text{and } X_i \to x_i \in P,\ 1 \leq i \leq n.
\end{aligned}$$

In the standard manner, let us extend $\Rightarrow$ to $\Rightarrow^m$, where $m \geq 0$; then, based on $\Rightarrow^m$, let us define $\Rightarrow^+$ and $\Rightarrow^*$. The language generated by G, denoted by $L(G)$, is defined as $L(G) = \{x : S \Rightarrow^* x,\ x \in T^*\}$. Let **$n$-PRLG** denote the language families of n-PRLGs.

2.4 CD Grammar Systems

A *general cooperating distributed grammar system* (a *general CD grammar system* for short) is a construct $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$, $n \geq 1$, where N is the alphabet of nonterminals, T is the alphabet of terminals, $N \cap T = \emptyset$, $S \in N$ is the start symbol, and, for $1 \leq i \leq n$, each component P_i is a finite set of general rules. (For the original context-free definition see [8].) For $u, v \in V^*$, $V = N \cup T$, and $1 \leq k \leq n$, let $u \Rightarrow_{P_k} v$ denote a derivation step performed by the application of a rule from P_k. As usual, let us extend the relation $\Rightarrow_{P_k}$ to $\Rightarrow^m_{P_k}$ (the m-step derivation), $m \geq 0$, $\Rightarrow^+_{P_k}$, and $\Rightarrow^*_{P_k}$. In addition, we define the relation $u \Rightarrow^t_{P_k} v$ so that $u \Rightarrow^*_{P_k} v$ and there is no $w \in V^*$ such that $v \Rightarrow_{P_k} w$. The language generated by Γ working in the f mode, $f \in \{*, t\}$, denoted by $L_f(\Gamma)$, is defined as $L_f(\Gamma) = \{w \in T^* : S \Rightarrow^f_{P_{k_1}} w_1 \Rightarrow^f_{P_{k_2}} \cdots \Rightarrow^f_{P_{k_l}} w_l = w,\ l \geq 1,\ 1 \leq k_i \leq n,\ 1 \leq i \leq l\}$. Γ is referred to as *rule-homogeneous*, *evenly rule-homogeneous*, or *context-free* (instead of general) if all its rules are homogeneous, evenly homogeneous, or context-free, respectively.

Language families generated by context-free CD grammar systems with n components working in the f mode and allowing ε-rules are denoted by $CD^{\varepsilon}_n(f)$. When the number of components is not limited, we replace n by ∞. The following results are well-known: (i) $CD^{\varepsilon}_{\infty}(*) = \mathbf{CF}$, (ii) $\mathbf{CF} = CD^{\varepsilon}_1(t) = CD^{\varepsilon}_2(t) \subset CD^{\varepsilon}_3(t) = CD^{\varepsilon}_{\infty}(t) = \mathbf{ET0L}$ (see Theorem 3.1 in [78]), where **ET0L** denotes the family of languages generated by extended tabled interactionless Lindenmayer systems (see [77]).

2.5 Finite Automata

There exist several different definitions of finite automata that are all widely used in formal language theory; but, they all have the same resulting accepting power. Notation-wise, one of the key differences between them is whether the definition uses a transition function or a set of transition rules. We present both of these notations since we work with jumping finite automata (see Section 2.6), which are based on lazy finite automata that use the set of transition rules, and we also work with Watson-Crick finite automata (see Section 2.7), which are based on finite automata that use the transition function.

2.5.1 Finite Automaton

A *finite automaton* is a quintuple $A = (V, Q, q_0, F, \delta)$, where V is an input alphabet, Q is a finite set of states, $V \cap Q = \emptyset$, $q_0 \in Q$ is the initial (or start) state, and $F \subseteq Q$ is a set of final (or accepting) states. The mapping δ is a transition function. If $\delta \colon Q \times (V \cup \{\varepsilon\}) \to 2^Q$, then the automaton is nondeterministic; if $\delta \colon Q \times V \to Q$, then the automaton is deterministic. A string w is accepted by a finite automaton if there is a sequence of transitions starting from q_0, ending in a state in F, and the symbols of the sequence yield w. A language is regular if and only if it can be recognized by a finite automaton.

2.5.2 Lazy Finite Automaton

A *lazy finite automaton* (see Section 2.6.2 in [91]), an *LFA* for short, is a quintuple $M = (Q, \Sigma, R, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $Q \cap \Sigma = \emptyset$, $R \subseteq Q \times \Sigma^* \times Q$ is finite, $s \in Q$ is the start state, and $F \subseteq Q$ is a set of final states. Members of R are referred to as rules of M. If $(p, y, q) \in R$ implies that $|y| \leq 1$, then M is a *finite automaton*, an *FA* for short. A configuration of M is any string in $Q\Sigma^*$. If $(p, y, q) \in R$ and

$x, y \in \Sigma^*$, then $pyx \Rightarrow qx$. In the standard manner, let us extend $\Rightarrow$ to $\Rightarrow^n$, where $n \geq 0$; then, based on $\Rightarrow^n$, let us define $\Rightarrow^+$ and $\Rightarrow^*$. The *language accepted by* M, denoted by $L(M)$, is defined as $L(M) = \{w \in \Sigma^* : sw \Rightarrow^* f,\ f \in F\}$. We say that M accepts w if and only if $w \in L(M)$. M rejects w if and only if $w \in \Sigma^* - L(M)$. Two LFAs M and M' are said to be equivalent if and only if $L(M) = L(M')$.

2.6 Jumping Finite Automata

A *general jumping finite automaton* (see [57, 58, 54]), a *GJFA* for short, is a quintuple $M = (Q, \Sigma, R, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $Q \cap \Sigma = \emptyset$, $R \subseteq Q \times \Sigma^* \times Q$ is finite, $s \in Q$ is the start state, and $F \subseteq Q$ is a set of final states. Members of R are referred to as rules of M. For brevity, we sometimes denote a rule (p, y, q) with a unique label h as $h : (p, y, q)$, and, instead of $h : (p, y, q) \in R$, we simply write $h \in R$. If $(p, y, q) \in R$ implies that $|y| \leq 1$, then M is a *jumping finite automaton*, a *JFA* for short. A configuration of M is any string in $\Sigma^* Q \Sigma^*$. The binary jumping relation, symbolically denoted by $\curvearrowright$ (or $\blacklozenge\curvearrowright$), over $\Sigma^* Q \Sigma^*$, is defined as follows. Let $x, z, x', z' \in \Sigma^*$ such that $xz = x'z'$ and $h : (p, y, q) \in R$; then, M makes a *jump* from $xpyz$ to $x'qz'$, symbolically written as $xpyz \curvearrowright x'qz'$ $[h]$, or simply $xpyz \curvearrowright x'qz'$. In the standard manner, let us extend $\curvearrowright$ to $\curvearrowright^n$, where $n \geq 0$; then, based on $\curvearrowright^n$, let us define $\curvearrowright^+$ and $\curvearrowright^*$. The language accepted by M, denoted by $L(M)$, is defined as $L(M) = \{uv : u, v \in \Sigma^*,\ usv \curvearrowright^* f,\ f \in F\}$.

We also recognize two special cases of the jumping relation as they are defined in [57, 58]. Let $M = (Q, \Sigma, R, s, F)$ be a GJFA. Let $w, x, y, z \in \Sigma^*$ and $(p, y, q) \in R$; then, (1) M makes a *left jump* from $wxpyz$ to $wqxz$, symbolically written as $wxpyz \ {}_l\curvearrowright wqxz$, and (2) M makes a *right jump* from $wpyxz$ to $wxqz$, symbolically written as $wpyxz \ {}_r\curvearrowright wxqz$. Let $u, v \in \Sigma^* Q \Sigma^*$; then, $u \curvearrowright v$ if and only if $u \ {}_l\curvearrowright v$ or $u \ {}_r\curvearrowright v$. Let us extend ${}_l\curvearrowright$ and ${}_r\curvearrowright$ to ${}_l\curvearrowright^n$, ${}_l\curvearrowright^*$, ${}_l\curvearrowright^+$, ${}_r\curvearrowright^n$, ${}_r\curvearrowright^*$ and ${}_r\curvearrowright^+$, where $n \geq 0$, by analogy with extending the corresponding notations for $\curvearrowright$. Set ${}_lL(M) = \{uv : u, v \in \Sigma^*,\ usv \ {}_l\curvearrowright^* f,\ f \in F\}$ and ${}_rL(M) = \{uv : u, v \in \Sigma^*,\ usv \ {}_r\curvearrowright^* f,\ f \in F\}$.

Let **GJFA**, **JFA**, ${}_l$**GJFA**, and ${}_r$**GJFA** denote the language families accepted by GJFAs, JFAs, GJFAs using only left jumps, and GJFAs using only right jumps, respectively.

2.7 $5' \to 3'$ Watson-Crick Finite Automata

In this part we recall some well-known concepts of DNA computing and related formal language theory. Readers who are not familiar with these topics can read [72].

Let V be an alphabet and $\rho \subseteq V \times V$ be a binary symmetric relation called complementarity. As the most prominent example, $V = \{A, C, G, T\}$ is used in DNA computing together with the Watson-Crick complementarity relation $\{(T, A), (A, T), (C, G), (G, C)\}$. The sequences built up by complementary pairs of letters are called double strands (of DNA).

A *Watson-Crick finite automaton* (or shortly, a *WK automaton*) is a finite automaton working on a Watson-Crick tape, that is, a double-stranded string (or molecule) in which the lengths of the strands are equal and the elements of the strands are pairwise complementary: $\left[\begin{smallmatrix} a_1 \\ b_1 \end{smallmatrix}\right]\left[\begin{smallmatrix} a_2 \\ b_2 \end{smallmatrix}\right] \cdots \left[\begin{smallmatrix} a_n \\ b_n \end{smallmatrix}\right] = \left[\begin{smallmatrix} a_1 a_2 \cdots a_n \\ b_1 b_2 \cdots b_n \end{smallmatrix}\right]$ where $a_i, b_i \in V$ and $(a_i, b_i) \in \rho$ for all $i = 1, \ldots, n$. The notation $\left[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right]$ is used only for strings $w_1, w_2 \in V^*$ with equal length and satisfying the complementarity relation ρ. The set of all double-stranded strings with this property is denoted by $\mathrm{WK}_\rho(V)$. For double-stranded strings for which these conditions are not

necessarily satisfied, the notation $\binom{w_1}{w_2}$ is used throughout the thesis. Formally, a WK automaton is $M = (V, \rho, Q, q_0, F, \delta)$, where V, Q, q_0, and F are the same as in finite automata, $\rho \subseteq V \times V$ is a symmetric relation (of complementarity), and the mapping $\delta\colon (Q \times \binom{V^*}{V^*}) \to 2^Q$ is a transition function such that $\delta(q, \binom{w_1}{w_2}) \neq \emptyset$ only for finitely many triples $(q, w_1, w_2) \in Q \times V^* \times V^*$.

The elementary difference between finite automata and WK automata, besides the doubled tape, is the number of heads. WK automata scan each of the two strands of the tape separately with a unique head. In classical WK automata, the heads scan both strands from left to right, and the processing of the input sequence ends when all complementary pairs of the sequence are read with both heads. Let $LM(M)$ denote the set of all double-stranded strings from $\mathrm{WK}_\rho(V)$ accepted by M. Let $\uparrow_V (LM(M)) = \{w_1 \in V^* : [\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}] \in LM(M),\ w_2 \in V^*\}$. Then, the language accepted by M, denoted $L(M)$, is defined as $L(M) = \uparrow_V (LM(M))$.

There are also some restricted variants of WK automata which are widely used in the literature (see, e.g., [72]):

- **N** : stateless, i.e., with only one state: if $Q = F = \{q_0\}$;
- **F** : all-final, i.e., with only final states: if $Q = F$;
- **S** : simple (at most one head moves in a step)
  $\delta\colon (Q \times ((\binom{V^*}{\{\varepsilon\}}) \cup (\binom{\{\varepsilon\}}{V^*}))) \to 2^Q$;
- **1** : 1-limited (exactly one letter is being read in a step)
  $\delta\colon (Q \times ((\binom{V}{\{\varepsilon\}}) \cup (\binom{\{\varepsilon\}}{V}))) \to 2^Q$.

Further variants such as **NS**, **FS**, **N1**, and **F1** WK automata can be identified in a straightforward way by combining multiple constraints.

In $5' \to 3'$ *WK automata* (see [60, 61, 63, 65, 70]), both heads start from the biochemical $5'$ end of the appropriate strand. Physically/mathematically and from a computing point of view they read the double-stranded sequence in opposite directions, while biochemically they go in the same direction. A $5' \to 3'$ WK automaton is *sensing* if the heads sense that they are meeting (i.e., they are close enough to meet in the next step or there is a possibility to read strings at overlapping positions). In sensing $5' \to 3'$ WK automata, the processing of the input sequence ends when for each pair of the sequence precisely one of the letters is read. Since the original Watson-Crick complementarity (in biology) is not only symmetric but also a one-to-one relation, we consider the input sequence to be fully processed, and thus the automaton makes a decision on the acceptance. Actually, it is a very natural assumption/restriction for most of the $5' \to 3'$ WK automata models that ρ defines a bijection on V.

In WK automata, the state transition δ is usually a mapping of the form $(Q \times \binom{V^*}{V^*}) \to 2^Q$. To help define an extended state transition δ' for sensing $5' \to 3'$ WK automata, in the transition $q' \in \delta(q, \binom{w_1}{w_2})$, we call $r_l = |w_1|$ and $r_r = |w_2|$ the left and right *radius* of the transition (they are the lengths of the strings that the heads will read from *left to right* and from *right to left* in this step, respectively). The value $r = r_l + r_r$ is the radius of the transition. Since $\delta(q, \binom{w_1}{w_2})$ is nonempty only for finitely many triples (q, w_1, w_2), there is a transition (maybe more) with the maximal radius for a given automaton. We extend δ to δ' with a sensing condition in the following way: Let $r_{\max}$ be the maximal radius among all rules. Then, let $\delta'\colon (Q \times \binom{V^*}{V^*} \times D) \to 2^Q$, where D is the *sensing distance*

set $\{-\infty, 0, 1, \ldots, r_{\max}, +\infty\}$. This set gives the distance of the two heads between 0 and $r_{\max}$, $+\infty$ when the heads are further than $r_{\max}$, or $-\infty$ when the heads are after their meeting point. Trivially, this automaton is finite, and D can be used only to control the sensing (i.e., the appropriate meeting of the heads). To describe the work of the automata, we use the concept of configuration. A configuration $\binom{w_1}{w_2}(q,s)\binom{w_1'}{w_2'}$ consists of the state q, the current sensing distance s, and the input $\left[\begin{smallmatrix} w_1w_1' \\ w_2w_2' \end{smallmatrix}\right] \in \mathrm{WK}_\rho(V)$ in such a way that the first head (on the upper strand) has already processed the part w_1, while the second head (on the lower strand) has already processed w_2'. A step of the sensing $5' \to 3'$ WK automaton, according to the state transition function δ', can be of the following two types:

(1) Normal steps : $\binom{w_1}{w_2y}(q,+\infty)\binom{xw_1'}{w_2'} \Rightarrow \binom{w_1x}{w_2}(q',s)\binom{w_1'}{yw_2'}$,
for $w_1, w_2, w_1', w_2', x, y \in V^*$ with $|w_2y| - |w_1| > r_{\max}$, $q, q' \in Q$,
if $\left[\begin{smallmatrix} w_1xw_1' \\ w_2yw_2' \end{smallmatrix}\right] \in \mathrm{WK}_\rho(V)$ and $q' \in \delta'(q, \binom{x}{y}, +\infty)$,
and $s = \begin{cases} |w_2| - |w_1x| & \text{if } |w_2| - |w_1x| \leq r_{\max}; \\ +\infty & \text{in other cases.} \end{cases}$

(2) Sensing steps : $\binom{w_1}{w_2y}(q,s)\binom{xw_1'}{w_2'} \Rightarrow \binom{w_1x}{w_2}(q',s')\binom{w_1'}{yw_2'}$,
for $w_1, w_2, w_1', w_2', x, y \in V^*$ and $s \in \{0, 1, \ldots, r_{\max}\}$ with $s = |w_2y| - |w_1|$,
if $\left[\begin{smallmatrix} w_1xw_1' \\ w_2yw_2' \end{smallmatrix}\right] \in \mathrm{WK}_\rho(V)$ and $q' \in \delta'(q, \binom{x}{y}, s)$,
and $s' = \begin{cases} s - |x| - |y| & \text{if } s - |x| - |y| \geq 0; \\ -\infty & \text{in other cases.} \end{cases}$

Note that there are no possible steps for the sensing distance $-\infty$. In the standard manner, let us extend $\Rightarrow$ to $\Rightarrow^n$, where $n \geq 0$; then, based on $\Rightarrow^n$, let us define $\Rightarrow^+$ and $\Rightarrow^*$. The set of all accepted double-stranded strings from $\mathrm{WK}_\rho(V)$, denoted by $LM(M)$, can be defined by the final accepting configurations that can be reached from the initial one: A double-stranded string $\left[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right] \in \mathrm{WK}_\rho(V)$ is accepted by a sensing $5' \to 3'$ WK automaton M if and only if $\binom{\varepsilon}{w_2}(q_0, s_0)\binom{w_1}{\varepsilon} \Rightarrow^* \left[\begin{smallmatrix} w_1' \\ w_2' \end{smallmatrix}\right](q_f, 0)\left[\begin{smallmatrix} w_1'' \\ w_2'' \end{smallmatrix}\right]$, for $q_f \in F$, where $\left[\begin{smallmatrix} w_1' \\ w_2' \end{smallmatrix}\right]\left[\begin{smallmatrix} w_1'' \\ w_2'' \end{smallmatrix}\right] = \left[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right]$ with the proper value of s_0 (it is $+\infty$ if $|w_1| > r_{\max}$, elsewhere it is $|w_1|$). Then, the language accepted by M, denoted $L(M)$, is defined as $L(M) = \uparrow_V (LM(M))$.

Besides the sensing version, papers [60, 61, 63, 65] also define the *full-reading sensing version*. The formal definition remains almost identical, however, the automaton continues with the reading after the heads met, and both heads have to read their strand completely from the $5'$ end to the $3'$ end. Therefore, this model actually defines the remaining steps for the sensing distance $-\infty$. The resulting behavior then combines some properties of classical WK automata and sensing $5' \to 3'$ WK automata. It can be easily seen that the full-reading sensing version is generally stronger than the sensing version. And finally, paper [70] introduces a new version of sensing $5' \to 3'$ WK automata without the sensing distance. It shows that it is not strictly necessary to know the precise sensing distance and that we can obtain the same power even if we are able to recognize only the actual meeting event of heads. Nonetheless, this result does not hold in general if we consider restricted variants of these models.

Part II

New Results on Jumping Automata

I'd like to have an argument, please.
—Monty Python's Flying Circus

Chapter 3

n-Parallel Jumping Finite Automata

This chapter covers our first steps to explore the possibilities of parallel jumping finite automata. The content of this chapter is composed of results that were presented at the conferences Excel@FIT 2015 (see [32]) and SDOT 2015 (see [37]) and also a few additional unpublished results; all written jointly with Alexander Meduna.

In terms of preliminaries, the reader should be familiar with the definitions of general notions (see Section 2.1), n-parallel right-linear grammars (see Section, 2.3), and jumping finite automata (see Section 2.6).

3.1 Introduction

In the previous century, most formal models were designed for continuous information processing. This, however, does not often reflects the requirements of modern information methods. Therefore, there is currently an active research around formal models that process information in a discontinuous way. Most notably, there are newly invented *jumping finite automata* (see [57, 58, 54]) that are completely focused on discontinuous reading. These automata go so far that they cannot even define some quite simple languages (e.g., a^*b^*) because they cannot guarantee any specific reading order between their jumps.

This chapter proposes a modification of these automata—*n-parallel jumping finite automata*. This modification presents a concept where the input is divided into several arbitrary parts and these parts are then separately processed with distinct synchronized heads. A quite similar concept was thoroughly studied in terms of formal grammars, where several nonterminals are being synchronously rewritten at once; for example, simple matrix grammars (see [25]) and n-parallel grammars (see [75, 90, 88, 73, 74]). However, to the best of our knowledge, no such research was done in terms of automata, where n heads synchronously read from distinct parts on the single tape. When this concept is combined with the mechanics of jumping finite automata, each part can be read discontinuously, but the overall order between parts is preserved; such an automaton then can handle additional languages (e.g., a^*b^*). Therefore, this modification represents the combined model of discontinuous and continuous reading.

The unrestricted version of jumping finite automata accepts a quite unique language family which initially had no known counterparts in grammars until jumping grammars were introduced (see [31]). Therefore, we have decided to base our initial research mainly

on the restricted version of these automata which use only right jumps. Note that restricted jumping finite automata define the same language family as classical finite automata. However, when such a restriction is combined with the previously described concept, we get a model which is very similar to n-parallel grammars. These automata utilize the jumping only during the initialization, when the heads jump to their start positions. After that, all heads read their parts of the input continuously in a left-to-right way. We compare these automata with n-parallel right-linear grammars and show that these models actually represent the same language families.

3.2 Definitions

In this section, we define a modification of jumping finite automata—n-parallel jumping finite automata—which read input words discontinuously with multiple synchronized heads. Moreover, we also define a more restricted mode for these automata which uses only the right jumps.

Definition 3.2.1. For $n \geq 1$, an *n-parallel general jumping finite automaton*, an *n-PGJFA* for short, is a quintuple

$$M = (Q, \Sigma, R, S, F),$$

where Q is a finite set of states, Σ is an input alphabet, $Q \cap \Sigma = \emptyset$, $R \subseteq Q \times \Sigma^* \times Q$ is finite, $S \subseteq Q^n$ is a set of start state strings, and $F \subseteq Q$ is a set of final states. Members of R are referred to as rules of M and instead of $(p, y, q) \in R$, we write $py \to q \in R$.

A *configuration* of M is any string in $\Sigma^* Q \Sigma^*$. Let X denote the set of all configurations over M. The binary jumping relation, symbolically denoted by $\curvearrowright$, over X, is defined as follows. Let $x, z, x', z' \in \Sigma^*$ such that $xz = x'z'$ and $py \to q \in R$; then, M makes a *jump* from $xpyz$ to $x'qz'$, symbolically written as

$$xpyz \curvearrowright x'qz'.$$

Let \$ be a special symbol, $\$ \notin (Q \cup \Sigma)$. An *$n$-configuration* of M is any string in $(X\{\$\})^n$. Let ${}_nX$ denote the set of all n-configurations over M. The binary n-jumping relation, symbolically denoted by ${}_n\curvearrowright$, over ${}_nX$, is defined as follows. Let $\zeta_1\$ \cdots \zeta_n\$, \vartheta_1\$ \cdots \vartheta_n\$ \in {}_nX$, so $\zeta_i, \vartheta_i \in X$, $1 \leq i \leq n$; then, M makes an *n-jump* from $\zeta_1\$ \cdots \zeta_n\$$ to $\vartheta_1\$ \cdots \vartheta_n\$$, symbolically written as

$$\zeta_1\$ \cdots \zeta_n\$ \ {}_n\curvearrowright \ \vartheta_1\$ \cdots \vartheta_n\$$$

if and only if $\zeta_i \curvearrowright \vartheta_i$ for all $1 \leq i \leq n$. In the standard manner, we extend ${}_n\curvearrowright$ to ${}_n\curvearrowright^m$, where $m \geq 0$. Let ${}_n\curvearrowright^+$ and ${}_n\curvearrowright^*$ denote the transitive closure of ${}_n\curvearrowright$ and transitive-reflexive closure of ${}_n\curvearrowright$, respectively.

The language accepted by M, denoted by $L(M, n)$, is defined as

$$\begin{aligned} L(M, n) = \{u_1 v_1 \cdots u_n v_n : u_1 s_1 v_1\$ \cdots u_n s_n v_n\$ \ {}_n\curvearrowright^* \ f_1\$ \cdots f_n\$, \\ u_i, v_i \in \Sigma^*, \ s_1 \cdots s_n \in S, \ f_i \in F, \ 1 \leq i \leq n\}. \end{aligned}$$

Let $w \in \Sigma^*$. We say that M accepts w if and only if $w \in L(M, n)$. M rejects w if and only if $w \in \Sigma^* - L(M, n)$.

Definition 3.2.2. For $n \geq 1$, let $M = (Q, \Sigma, R, S, F)$ be an n-PGJFA, and let X denote the set of all configurations over M. The binary right jumping relation, symbolically denoted by ${}_r\curvearrowright$, over X, is defined as follows. Let $w, x, y, z \in \Sigma^*$, and $py \to q \in R$; then, M makes a *right jump* from $wpyxz$ to $wxqz$, symbolically written as

$$wpyxz \ {}_r\curvearrowright \ wxqz.$$

Let ${}_nX$ denote the set of all n-configurations over M. The binary right n-jumping relation, symbolically denoted by ${}_{n-r}\curvearrowright$, over ${}_nX$, is defined as follows. Let $\zeta_1\$\cdots\zeta_n\$, \vartheta_1\$\cdots\vartheta_n\$ \in {}_nX$, so $\zeta_i, \vartheta_i \in X$, $1 \leq i \leq n$; then, M makes a *right n-jump* from $\zeta_1\$\cdots\zeta_n\$$ to $\vartheta_1\$\cdots\vartheta_n\$$, symbolically written as

$$\zeta_1\$\cdots\zeta_n\$ \ {}_{n-r}\curvearrowright \ \vartheta_1\$\cdots\vartheta_n\$$$

if and only if $\zeta_i \ {}_r\curvearrowright \vartheta_i$ for all $1 \leq i \leq n$.

We extend ${}_{n-r}\curvearrowright$ to ${}_{n-r}\curvearrowright^m$, ${}_{n-r}\curvearrowright^+$, and ${}_{n-r}\curvearrowright^*$, where $m \geq 0$, by analogy with extending the corresponding notations for ${}_n\curvearrowright$. Let $L(M, n{-}r)$ denote the language accepted by M using only right n-jumps.

Let **n-PGJFA** and **${}_rn$-PGJFA** denote the language families accepted by n-PGJFAs and n-PGJFAs using only right n-jumps, respectively.

3.3 Examples

To demonstrate the behavior of these automata, we present two simple examples.

Example 3.3.1. Consider the 2-PGJFA

$$M = (\{s, r, p, q\}, \Sigma, R, \{sr\}, \{s, r\}),$$

where $\Sigma = \{a, b, c, d\}$ and R consists of the rules

$$sa \to p, \ \ pb \to s, \ \ rc \to q, \ \ qd \to r.$$

Starting from sr, M has to read some a and b with the first head and some c and d with the second head, entering again the start (and also the final) states sr. If we work with the unrestricted jumps, both heads can read the symbols in an arbitrary order. However, if we work with the right jumps, both heads must read all symbols in their original order; otherwise, the automaton will eventually get stuck. Therefore, the accepted languages are

$$L(M, 2) = \{uv : u \in \{a, b\}^*, \ v \in \{c, d\}^*, \ |u|_a = |u|_b = |v|_c = |v|_d\}.$$
$$L(M, 2{-}r) = \{(ab)^n(cd)^n : n \geq 0\}$$

Example 3.3.2. Consider the 2-PGJFA

$$M = (\{s, r, t\}, \Sigma, R, \{ss\}, \{s\}),$$

where $\Sigma = \{a, b, c\}$ and R consists of the rules

$$sa \to r, \ \ rb \to t, \ \ tc \to s.$$

Starting from ss, M has to read some a, b, and c with both heads, entering again the start (and also the final) states ss. Therefore, the accepted languages are

$$L(M, 2) = \{uv : u, v \in \{a, b, c\}^*, \ |u|_a = |u|_b = |u|_c = |v|_a = |v|_b = |v|_c\},$$
$$L(M, 2{-}r) = \{uu : u \in \{abc\}^*\}.$$

It can be easily shown that the languages accepted with unrestricted n-jumps in Examples 3.3.1 and 3.3.2 cannot be defined by any original jumping finite automata. In the case of languages accepted with right n-jumps, Example 3.3.1 defines a linear language, but Example 3.3.2 defines only a regular language.

3.4 Unrestricted n-Jumping Relation

This section gives a basic characterization of the language families accepted by n-PGJFAs with unrestricted n-jumps. Most notably, we show that n-PGJFAs with unrestricted n-jumps define an infinite hierarchy of language families.

Theorem 3.4.1. *1-PGJFA = GJFA.*

Proof. The definition of the binary jumping relation is identical between GJFAs (see Section 2.6) and n-PGJFAs (see Definition 3.2.1). Consequently, if $n = 1$, both models transit between configurations in the same way, and they also require the same conditions for accepting configurations. Therefore, the only difference is in their initial configurations since GJFAs have a single start state but 1-PGJFAs have a set of start states. Nonetheless, we can convert any 1-PGJFA $M = (Q, \Sigma, R, S, F)$ into the equivalent 1-PGJFA $N = (Q', \Sigma, R', \{s\}, F)$ such that $s \notin (Q \cup \Sigma)$, $Q' = Q \cup \{s\}$, and $R' = R \cup \{s \to s' : s' \in S\}$. Then, the conversions between GJFAs and 1-PGJFAs are trivial. □

Lemma 3.4.2. *For all $n \geq 1$, there is an n-PGJFA $M = (Q, \Sigma, R, S, F)$ such that $\Sigma = \{a_1, \ldots, a_n\}$ and $L(M, n) = \{a_1\}^* \cdots \{a_n\}^*$.*

Proof. By construction. For any $n \geq 1$, define the n-PGJFA $M = (Q, \Sigma, R, S, F)$, where $Q = \{s_1, \ldots, s_n\}$, $\Sigma = \{a_1, \ldots, a_n\}$, $R = \{s_i a_i \to s_i,\ s_i \to s_i : 1 \geq i \geq n\}$, $S = \{s_1 \cdots s_n\}$, and $F = Q$. Observe that each head handles a different symbol and that it can read zero or one occurrence of this symbol in each step. Therefore, the accepted language is clearly $L(M, n) = \{a_1\}^* \cdots \{a_n\}^*$. □

Lemma 3.4.3. *For all $n \geq 1$ and $m > n$, there is no n-PGJFA $M = (Q, \Sigma, R, S, F)$ such that $\Sigma = \{a_1, \ldots, a_m\}$ and $L(M, n) = \{a_1\}^* \cdots \{a_m\}^*$.*

Proof. We extend the reasoning from Lemma 19 in [57] that shows that there is no GJFA that accepts $\{a\}^*\{b\}^*$. By contradiction. Assume that, for some $n \geq 1$ and $m > n$, there is a n-PGJFA $M = (Q, \Sigma, R, S, F)$ such that $\Sigma = \{a_1, \ldots, a_m\}$ and $L(M, n) = \{a_1\}^* \cdots \{a_m\}^*$. Then, some of the heads of M must handle at least two types of symbols. Assume any $w_1 uvw_2 \in L(M, n)$ such that uv is a whole part read with one head, uv contains at least two types of symbols, v is read in a single step, $|u| \geq 1$, and $|v| \geq 1$. Clearly, for any $n \geq 1$ and $m > n$, there has to exist some $w_1 uvw_2 \in L(M, n)$ that satisfies these conditions. Due to the behavior of the unrestricted n-jumps, it must then also hold that $w_1 vuw_2 \in L(M, n)$; however, $w_1 vuw_2 \notin \{a_1\}^* \cdots \{a_m\}^*$. That is a contradiction with the assumption that $L(M, n) = \{a_1\}^* \cdots \{a_m\}^*$. Therefore, there is no n-PGJFA $M = (Q, \Sigma, R, S, F)$ such that $n \geq 1$, $m > n$, $\Sigma = \{a_1, \ldots, a_m\}$, and $L(M, n) = \{a_1\}^* \cdots \{a_m\}^*$. □

Theorem 3.4.4. *For all $n \geq 1$, n-PGJFA $\subset$ $(n+1)$-PGJFA.*

Proof. First, we show that, for all $n \geq 1$, **n-PGJFA $\subseteq$ $(n+1)$-PGJFA**. For any n-PGJFA $M = (Q, \Sigma, R, S, F)$ we can construct the $(n+1)$-PGJFA $N = (Q', \Sigma, R', S', F')$ such that $f \notin (Q \cup \Sigma)$, $Q' = Q \cup \{f\}$, $R' = R \cup \{f \to f\}$, $S' = \{wf : w \in S\}$, $F' = F \cup \{f\}$. It

is not hard to see that $L(M,n) = L(N,n+1)$. Second, for all $n \geq 1$, $(n+1)$-**PGJFA** $\not\subseteq$ n-**PGJFA** follows directly from Lemmas 3.4.2 and 3.4.3. □

With Theorems 3.4.1 and 3.4.4 we can easily derive the following additional characterization of the language families accepted by n-PGJFAs with unrestricted n-jumps.

Theorem 3.4.5. *For all $n \geq 1$, **FIN** $\subset$ n-**PGJFA**.*

Proof. This theorem directly follows from **FIN** $\subset$ **GJFA** (see [57]). □

Theorem 3.4.6. *For all $n \geq 1$, n-**PGJFA** $\not\subseteq$ **REG** and n-**PGJFA** $\not\subseteq$ **CF**.*

Proof. This theorem directly follows from the fact that there is a GJFA $M = (Q, \Sigma, R, s, F)$ such that $\Sigma = \{a, b, c\}$ and $L(M) = \{w \in \Sigma^* : |w|_a = |w|_b = |w|_c\}$ which is a well-known non-context-free language (see [57]). □

Theorem 3.4.7. *For all $n \geq 1$, n-**PGJFA** $\subset$ **CS**.*

Proof. The jumps of GJFAs can be simulated by linear bounded automata (see [57]), and the same also holds for the n-jumps of n-PGJFAs. Thus, for all $n \geq 1$, n-**PGJFA** $\subseteq$ **CS**. From Lemma 3.4.3, for all $n \geq 1$, **CS** $-$ n-**PGJFA** $\neq \emptyset$. □

3.5 Right n-Jumping Relation

This section gives a detailed characterization of the language families accepted by n-PGJFAs with right n-jumps. First, we prove that n-PGJFAs with right n-jumps and n-PRLGs define the same language families.

Lemma 3.5.1. *For every n-PRLG $G = (N_1, \ldots, N_n, T, S1, P)$, there is an n-PGJFA $M = (Q, \Sigma, R, S2, F)$ using only right n-jumps such that $L(M, n{-}r) = L(G)$.*

Proof. Let $G = (N_1, \ldots, N_n, T, S1, P)$ be an n-PRLG. Without loss of generality, assume that $f \notin (N_1 \cup \cdots \cup N_n \cup T)$. Keep the same n and define the n-PGJFA

$$M = (\{f\} \cup N_1 \cup \cdots \cup N_n, T, R, S2, \{f\}),$$

where R and $S2$ are constructed in the following way:

(1) For each rule of the form $S1 \to X_1 \cdots X_n$, $X_i \in N_i$, $1 \leq i \leq n$, in P, add the start state string $X_1 \cdots X_n$ to $S2$.
(2) For each rule of the form $X \to aY$, $X, Y \in N_i$, for some i, $1 \leq i \leq n$, $a \in T^*$, in P, add the rule $Xa \to Y$ to R.
(3) For each rule of the form $X \to a$, $X \in N_i$, for some i, $1 \leq i \leq n$, $a \in T^*$, in P, add the rule $Xa \to f$ to R.

Observe that the constructed n-PGJFA M with right n-jumps simulates the n-PRLG G in such a way that its heads read symbols in the same fashion as the nonterminals of G generate them.

Any sentence $w \in L(G)$ can be divided into $w = u_1 \cdots u_n$, where u_i represents the part of the sentence which can be generated from the nonterminal X_i of a rule $S1 \to X_1 \cdots X_n$, $X_i \in N_i$, $1 \leq i \leq n$. In the same way, M can start from an n-configuration $X_1 u_1 \$ \cdots X_n u_n \$$, where the heads with the states X_i have to read u_i. Therefore the part (1), where we convert the rules $S1 \to X_1 \cdots X_n$ into the start state strings, and the selection of a start state string thus covers the first derivation step of the grammar.

Any consecutive non-final derivation step of the grammar then rewrites all n nonterminals in the sentential form with the rules of the form $X \to aY$, $X, Y \in N_i$, for some i, $1 \leq i \leq n$, $a \in T^*$. Therefore the part (2), where we convert the grammar rules $X \to aY$ into the automaton rules $Xa \to Y$. The automaton M always works with all its heads simultaneously, and thus the equivalent effect of these steps should be obvious.

In the last derivation step of the grammar, every nonterminal is rewritten with a rule of the form $X \to a$, $X \in N_i$, for some i, $1 \leq i \leq n$, $a \in T^*$. We can simulate the same behavior in the automaton if we end up in a final state from which there are no ongoing rules. Therefore the part (3), where we convert the grammar rules $X \to a$ into the automaton rules $Xa \to f$, where f is the sole final state. All heads of the automaton must also end up in this final state simultaneously, or the automaton will get stuck; there are no ongoing rules from f, and all heads must make a move during every step.

The automaton M can also start from an n-configuration where the input is divided into such parts that they cannot be generated from the nonterminals X_i of the rules $S1 \to X_1 \cdots X_n$, $X_i \in N_i$, $1 \leq i \leq n$. However, such an attempt will eventually get the automaton stuck because M simulates only the derivation steps of the grammar. □

Lemma 3.5.2. *For every n-PGJFA $M = (Q, \Sigma, R, S2, F)$ using only right n-jumps, there is an n-PRLG $G = (N_1, \ldots, N_n, T, S1, P)$ such that $L(G) = L(M, n{-}r)$.*

Proof. Let $M = (Q, \Sigma, R, S2, F)$ be an n-PGJFA with right n-jumps. Keep the same n and define the n-PRLG

$$G = (N_1, \ldots, N_n, \Sigma, S1, P),$$

where $N_1, \ldots, N_n$, and P are constructed in the following way:

(1) For each state $p \in Q$,
add the nonterminal p_i to N_i for all $1 \leq i \leq n$.
(2) For each start state string $p_1 \cdots p_n \in S2$, $p_i \in Q$, $1 \leq i \leq n$,
add the start rule $S1 \to p_{1_1} \cdots p_{n_n}$ to P.
(3) For each rule $py \to q \in R$, $p, q \in Q$, $y \in \Sigma^*$,
add the rule $p_i \to yq_i$ to P for all $1 \leq i \leq n$.
(4) For each state $p \in F$,
add the rule $p_i \to \varepsilon$ to P for all $1 \leq i \leq n$.

Observe that the constructed n-PRLG G simulates the n-PGJFA M with right n-jumps in such a way that its nonterminals generate terminals in the same fashion as the heads of M read them.

The definition of n-PRLGs requires that $N_1, \ldots, N_n$ are mutually disjoint nonterminal alphabets. However, the states of n-PGJFAs do not have such a restriction. Therefore, we use a new index in all converted occurrences of states, this creates a separate item for each nonterminal position. The index is represented by i and is used in all conversion steps.

Any sentence $w \in L(M, n{-}r)$ can be divided into $w = u_1 \cdots u_n$, where u_i represents the part of the sentence which can be accepted by the head of M with a start state p_i from a start n-configuration $p_1 u_1 \$ \cdots p_n u_n \$$, where $p_1 \cdots p_n \in S2$, $1 \leq i \leq n$. In the grammar, we can simulate the start n-configurations with the rules $S1 \to p_{1_1} \cdots p_{n_n}$, where the nonterminals p_{i_i} must be able to generate u_i. Therefore the part (2), where we convert the start state strings into the rules.

During every step of the automaton all heads simultaneously make a move. Likewise, during every non-initial step of the grammar all non-terminals are simultaneously rewritten. Therefore the part (3), where we convert the automaton rules $py \to q$ into the grammar rules $p_i \to yq_i$. The equivalent effect of these steps should be obvious.

The automaton can successfully end if all its heads are in the final states. We can simulate this situation in the grammar if we rewrite every nonterminal to ε. Therefore the part (4), where we create new erasing rules for all final states. These rules can be used only once during the last derivation step of the grammar; otherwise, the generation process of the grammar will get stuck. □

Theorem 3.5.3. ***${}_r n$-PGJFA = n-PRLG.***

Proof. **n-PRLG $\subseteq {}_r n$-PGJFA** follows from Lemma 3.5.1. **${}_r n$-PGJFA $\subseteq$ n-PRLG** follows from Lemma 3.5.2. □

With Theorem 3.5.3 we can easily derive the following additional characterization of the language families accepted by n-PGJFAs using only right n-jumps.

Theorem 3.5.4. *For all $n \geq 1$, ${}_r n$-PGJFA $\subset {}_r(n+1)$-PGJFA.*

Proof. This theorem directly follows from **n-PRLG $\subset$ $(n+1)$-PRLG** (see [75]). □

Theorem 3.5.5. *For all $n \geq 1$, ${}_r n$-PGJFA is closed under union, finite substitution, homomorphism, reflection, and intersection with a regular set.*

Proof. This theorem directly follows from the same results for **n-PRLG** (see [75]). □

Theorem 3.5.6. *For all $n \geq 2$, ${}_r n$-PGJFA is not closed under intersection or complement.*

Proof. This theorem directly follows from the same results for **n-PRLG** (see [75]). □

Theorem 3.5.7. ***${}_r 1$-PGJFA = ${}_r$GJFA = REG.***

Proof. This theorem directly follows from **1-PRLG = REG** (see [75]) and from **${}_r$GJFA = REG** (see [57]). □

Theorem 3.5.8. ***${}_r 2$-PGJFA $\subset$ CF.***

Proof. This theorem directly follows from **2-PRLG $\subset$ CF** (see [75]). □

Theorem 3.5.9. ***${}_r n$-PGJFA $\subset$ CS** and there exist non-context-free languages in ${}_r n$-PGJFA for all $n \geq 3$.*

Proof. This theorem directly follows from the same results for **n-PRLG** (see [75]). □

3.6 Concluding Remarks

The presented results show that the concept of the parallel jumping has a positive effect on the model of jumping finite automata. The most significant part of these results is the fact that every additional head always increases the power of these automata, and this is true for both the unrestricted and right n-jumping relation. Therefore, this creates two infinite hierarchies of language families. Next, due to the very simple conversions and similar concepts, we can see n-parallel general jumping finite automata using only right n-jumps as a direct counterpart to n-parallel right-linear grammars. There are already other automata with the same power as n-parallel right-linear grammars (e.g., self-regulating finite automata, see [52]), however, they use considerably different mechanisms and the

conversions between models are not straightforward; therefore, they would hardly qualify as a direct counterpart. Furthermore, with a little bit of tweaking, we could easily adjust our model so that it coincides with other well-known grammars that synchronously rewrite several nonterminals at once, e.g., right-linear simple matrix grammars (see [25]). On the other hand, considering n-parallel general jumping finite automata with unrestricted n-jumps, due to their unconventional behavior we were not able to find a fitting counterpart for them in grammars. It is possible that no such a counterpart exists and it would need to be introduced; as it was with general jumping finite automata and their counterpart in the form of jumping grammars (see [31]).

Finally, we propose some suggestions for further investigation.

(I) In this chapter, we have considered only the situation where all heads keep their own state and always work synchronously together. Investigate other options, where, e.g., all heads follow a single state, not all heads have to make a move during the n-jump, or only one head can make a move during the n-jump.

(II) Consider other grammars that rewrite several nonterminals at once for which there is no direct counterpart in the form of an automaton model. Can we use the concept of the parallel jumping to introduce such a model?

(III) Study closure and decidability properties for n-parallel general jumping finite automata with unrestricted n-jumps.

May I see your silly walk?
—Monty Python's Flying Circus

Chapter 4

Double-Jumping Finite Automata

This chapter studies the advanced possibilities of the two-head jumping finite automaton model under various reading modes. The content of this chapter is composed of results that were published at the conference NCMA 2016 (see [33]) and in the journal RAIRO (see [34]); all written jointly with Zbyněk Křivka and Alexander Meduna.

In terms of preliminaries, the reader should be familiar with the definitions of general notions (see Section 2.1), grammars and endmarking (see Sections 2.2.1, 2.2.2, 2.2.3), lazy finite automata (see Section 2.5.2), and jumping finite automata (see Section 2.6).

4.1 Introduction

At present, jumping versions of formal models, such as grammars and automata, represent a vivid investigation area in formal language theory. The current chapter continues with this investigation in terms of jumping finite automata.

Consider the notion of a jumping finite automaton M as sketched in Section 1.2. This chapter modifies the way M works so it simultaneously performs two jumps according to the same rule. For either of the two jumps, it always considers three natural directions—(1) to the left, (2) to the right, and (3) in either direction.

In correspondence to this jumping-direction three-part classification, this chapter investigates the mutual relation between the language families resulting from jumping finite automata working in these ways and the families of regular, linear, context-free, and context-sensitive languages. In essence, it demonstrates that most of these language families are pairwise incomparable—that is, they are not subfamilies of each other and, simultaneously, they are not disjoint either.

In addition, this chapter also establishes several closure and non-closure properties concerning the language families defined by these jumping finite automata.

4.2 Definitions

Considering the definition of jumping finite automata from Section 2.6, we define a new mode for general jumping finite automata that performs two single jumps simultaneously. In this mode, both single jumps follow the same rule, however, they are performed on two different positions on the tape and thus handle different parts of the input string. Moreover, these two jumps cannot ever cross each other—their initial mutual order is preserved during the whole process. As a result, when needed, we can specifically denote them as the *first*

jump and the *second jump*. Furthermore, this chapter considers three possible types of single jumps that can be used in this new double-jumping mode. Besides the unrestricted single jump $\blacklozenge\curvearrowright$ from the original definition, we also define and use two restricted single jumps with limited movement. The definition of the restricted single jumps is modified from the original definition in order to get a more consistent behavior. The restricted single jumps now read strings from the configuration of an automaton on the specific side of their state depending on the actual direction of their jumping. (For example, if an automaton jumps to the left, it reads a string on the left of the current state.)

Let $M = (Q, \Sigma, R, s, F)$ be a GJFA. Let $w, x, y, z \in \Sigma^*$ and $h : (p, y, q) \in R$; then, $wpyxz \blacktriangleright\curvearrowright wxqz\ [h]$ and $wxypz \blacktriangleleft\curvearrowright wqxz\ [h]$ in M.

We combine two single jumps into the unrestricted and four types of restricted 2-jumping relations. The restricted relations are the main subject of this chapter, and we show in the next sections that these restrictions severely and uniquely impact the behavior of the automaton and thus also the families of accepted languages and their closure properties.

Let X denote the set of all configurations of M. A 2-*configuration* of M is any string in XX. Let X^2 denote the set of all 2-configurations of M. For brevity, let $t_1t_2 \in \{\blacklozenge\blacklozenge, \blacktriangleright\blacktriangleright, \blacktriangleright\blacktriangleleft, \blacktriangleleft\blacktriangleright, \blacktriangleleft\blacktriangleleft\}$ such that $t_1, t_2 \in \{\blacklozenge, \blacktriangleright, \blacktriangleleft\}$. The binary t_1t_2 2-*jumping relation*, symbolically denoted by ${}_{t_1t_2}\curvearrowright$, over X^2, is defined as follows. Let $\zeta_1\zeta_2, \vartheta_1\vartheta_2 \in X^2$, where $\zeta_1, \zeta_2, \vartheta_1, \vartheta_2 \in X$, and $h \in R$; then, M makes a t_1t_2 2-*jump* from $\zeta_1\zeta_2$ to $\vartheta_1\vartheta_2$ according to h, symbolically written as

$$\zeta_1\zeta_2 \ {}_{t_1t_2}\curvearrowright \ \vartheta_1\vartheta_2 \ [h]$$

if and only if $\zeta_1 \ {}_{t_1}\curvearrowright \vartheta_1\ [h]$ and $\zeta_2 \ {}_{t_2}\curvearrowright \vartheta_2\ [h]$. Depending on the specific type of jumps $\blacklozenge\blacklozenge$, $\blacktriangleright\blacktriangleright$, $\blacktriangleright\blacktriangleleft$, $\blacktriangleleft\blacktriangleright$, $\blacktriangleleft\blacktriangleleft$, we use the following naming: unrestricted, right-right, right-left, left-right, left-left 2-jumping relation (or 2-jump), respectively.

Let o be any of the jumping direct relations introduced above. In the standard way, we extend o to o^m, $m \geq 0$; o^+; and o^*. To express that M only performs jumps according to o, write M_o. If o is one of the relations $\blacklozenge\curvearrowright$, $\blacktriangleright\curvearrowright$, $\blacktriangleleft\curvearrowright$, set

$$L(M_o) = \{uv : u, v \in \Sigma^*,\ usv \ \ o^*\ f,\ f \in F\}.$$

If o is one of the relations $\blacklozenge\blacklozenge\curvearrowright$, $\blacktriangleright\blacktriangleright\curvearrowright$, $\blacktriangleright\blacktriangleleft\curvearrowright$, $\blacktriangleleft\blacktriangleright\curvearrowright$, $\blacktriangleleft\blacktriangleleft\curvearrowright$, set

$$L(M_o) = \{uvw : u, v, w \in \Sigma^*,\ usvsw \ \ o^*\ ff,\ f \in F\}.$$

$L(M_o)$ is referred to as the *language of* M_o. Set $\mathscr{L}_o = \{L(M_o) : M \text{ is a GJFA}\}$; $\mathscr{L}_o$ is referred to as the *language family accepted by GJFAs according to* o.

To illustrate this terminology, take $o = \blacklozenge\blacklozenge\curvearrowright$. Consider $M_{\blacklozenge\blacklozenge\curvearrowright}$. Notice that

$$L(M_{\blacklozenge\blacklozenge\curvearrowright}) = \{uvw : u, v, w \in \Sigma^*,\ usvsw \ \ \blacklozenge\blacklozenge\curvearrowright^*\ ff,\ f \in F\}.$$

$L(M_{\blacklozenge\blacklozenge\curvearrowright})$ is referred to as the *language of* $M_{\blacklozenge\blacklozenge\curvearrowright}$. Set $\mathscr{L}_{\blacklozenge\blacklozenge\curvearrowright} = \{L(M_{\blacklozenge\blacklozenge\curvearrowright}) : M \text{ is a GJFA}\}$; $\mathscr{L}_{\blacklozenge\blacklozenge\curvearrowright}$ is referred to as the *language family accepted by GJFAs according to* $\blacklozenge\blacklozenge\curvearrowright$.

Furthermore, set $\mathscr{L}_2 = \mathscr{L}_{\blacklozenge\blacklozenge\curvearrowright} \cup \mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright} \cup \mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright} \cup \mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright} \cup \mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$.

Lastly, we define an auxiliary subfamily of the family of regular languages that will be useful to the study of the accepting power of GJFAs that perform right-left and left-right 2-jumps. In Section 4.3.2, it helps us to describe the regular portions of the appropriate language families.

Definition 4.2.1. Let $L_{m,n}$ be a *simply-expandable language* (SEL) over an alphabet Σ if it can be written as follows. Let m and n be positive integers; then,

$$L_{m,n} = \bigcup_{h=1}^{m} \left\{ u_{h,1}u_{h,2}\cdots u_{h,n}v_h^i v_h^i u_{h,n}\cdots u_{h,2}u_{h,1} : i \geq 0,\ u_{h,k}, v_h \in \Sigma^*,\ 1 \leq k \leq n \right\}.$$

Let **SEL** denote the family of SELs. For the sake of clarity, let us note that, in the previous definition, v_h and all $u_{h,k}$ are fixed strings that only vary for different values of h.

4.3 General Results

This section studies the accepting power of GJFAs making their computational steps by unrestricted, right-left, left-right, right-right, and left-left 2-jumps.

4.3.1 Unrestricted 2-Jumping Relation

Example 4.3.1. Consider the GJFA

$$M_{\blacklozenge\blacklozenge\curvearrowright} = (\{s, f\}, \Sigma, R, s, \{f\}),$$

where $\Sigma = \{a, b, c\}$ and R consists of the rules (s, ab, f) and (f, c, f). Starting from s, M has to read two times some ab, entering the final state f; then, M can arbitrarily many times read two times some c. Consequently, if we work with the unrestricted 2-jumps, the input must always contain two separate strings ab, and the symbols c can be anywhere around these two strings. Therefore, the accepted language is

$$L(M_{\blacklozenge\blacklozenge\curvearrowright}) = \{c^k abc^m abc^n : k + m + n \text{ is an even integer},\ k, m, n \geq 0\}.$$

Lemma 4.3.2. *For every language $L \in \mathscr{L}_2$, there is no $x \in L$ such that $|x|$ is an odd number; furthermore, there is no symbol a for which $|x|_a$ is an odd number.*

Proof. By the definition of 2-jumps, any GJFA that uses 2-jumps always performs two single jumps simultaneously, and they both follow the same rule, therefore, there is no way how to read an odd number of symbols from the input string. □

Lemma 4.3.3. *There is no GJFA $M_{\blacklozenge\curvearrowright}$ that accepts $\{c^k abc^m abc^n : k + m + n$ is an even integer, $k, m, n \geq 0\}$.*

Proof. We follow Lemma 19 from [57] which effectively shows that a GJFA $M_{\blacklozenge\curvearrowright}$ can maintain a specific order of symbols only in the sole context of a rule. By contradiction. Let $K = \{c^k abc^m abc^n : k + m + n$ is an even integer, $k, m, n \geq 0\}$. Assume that there is a GJFA $M_{\blacklozenge\curvearrowright}$ such that $L(M_{\blacklozenge\curvearrowright}) = K$. If M uses two times a rule reading ab, then it can also accept input $aabb$; and clearly $aabb \notin K$. Consequently, M has to always read the whole sequence $abc^m ab$ with a single rule; however, the number m is unbounded, and thus there cannot be finitely many rules that cover all possibilities—a contradiction with the assumption that $L(M_{\blacklozenge\curvearrowright}) = K$ exists. Therefore, there is no GJFA $M_{\blacklozenge\curvearrowright}$ that accepts $\{c^k abc^m abc^n : k + m + n$ is an even integer, $k, m, n \geq 0\}$. □

Theorem 4.3.4. *$\mathscr{L}_{\blacklozenge\curvearrowright}$ and $\mathscr{L}_{\blacklozenge\blacklozenge\curvearrowright}$ are incomparable.*

Proof. $\mathscr{L}_{\blacklozenge\curvearrowright} \not\subseteq \mathscr{L}_{\blacklozenge\blacklozenge\curvearrowright}$ follows from **FIN** $\subset \mathscr{L}_{\blacklozenge\curvearrowright}$ (see Theorem 18 in [57]) and Lemma 4.3.2. $\mathscr{L}_{\blacklozenge\blacklozenge\curvearrowright} \not\subseteq \mathscr{L}_{\blacklozenge\curvearrowright}$ follows from Example 4.3.1 and Lemma 4.3.3. Moreover, both $\mathscr{L}_{\blacklozenge\curvearrowright}$ and $\mathscr{L}_{\blacklozenge\blacklozenge\curvearrowright}$ clearly contain the simple finite language $\{aa\}$. □

4.3.2 Right-Left 2-Jumping Relation

Claim 4.3.5. *Let $M = (Q, \Sigma, R, s, F)$ be a GJFA; then, every $x \in L(M_{\blacktriangleright\blacktriangleleft\curvearrowright})$ can be written as $x = u_1u_2 \cdots u_nu_n \cdots u_2u_1$, where $n \geq 1$, and $u_i \in \Sigma^*$, $1 \leq i \leq n$.*

Proof. Consider any GJFA $M_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q, \Sigma, R, s, F)$. Since we work with the right-left 2-jumps, the first jump can move only to the right, the second jump can move only to the left, and both jumps cannot cross each other. Observe that if the configuration of M is of the form $upvpw$, where $u, v, w \in \Sigma^*$, and $p \in Q$, then M cannot read the symbols in u and w anymore. Also, observe that this covers the situation when M starts to accept $x \in \Sigma^*$ from any other configuration than sxs. Therefore, to read the whole input string, M has to start in the configuration sxs, and it cannot jump over any symbols during the whole process. Consequently, since both jumps always follow the same rule, they have to read the same corresponding strings and, at the end of the process, meet in the middle of the input string. Therefore, every $x \in L(M_{\blacktriangleright\blacktriangleleft\curvearrowright})$ can be surely written as $x = u_1u_2 \cdots u_nu_n \cdots u_2u_1$, where $n \geq 1$, and $u_i \in \Sigma^*$, $1 \leq i \leq n$. □

Lemma 4.3.6. *For every GJFA M, there is a linear grammar G such that $L(M_{\blacktriangleright\blacktriangleleft\curvearrowright}) = L(G)$.*

Proof. Consider any GJFA $M_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q, \Sigma, R, s, F)$. Define the linear grammar $G = (Q, \Sigma, P, s)$, where P is constructed in the following way:
(1) For each $(p, y, q) \in R$, add $p \to yqy$ to P.
(2) For each $p \in F$, add $p \to \varepsilon$ to P.

We follow Claim 4.3.5 and its proof. Let $p, q \in Q$, $f \in F$, and $y, u, v, w \in \Sigma^*$. Observe that every time M can make a 2-jump $pywyp \blacktriangleright\blacktriangleleft\curvearrowright qwq$ according to $(p, y, q) \in P$, G can also make the derivation step $upv \Rightarrow uyqyv$ according to $p \to yqy \in P$. Moreover, every time M is in a final state f, G can finish the string with $f \to \varepsilon \in P$. Finally, observe that G cannot do any other action, therefore, $L(M_{\blacktriangleright\blacktriangleleft\curvearrowright}) = L(G)$. □

Theorem 4.3.7. $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright} \subset \boldsymbol{LIN}_{even}$.

Proof. $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright} \subseteq \mathbf{LIN}_{\text{even}}$ follows from Lemma 4.3.6 and the structure of its proof. $\mathbf{LIN}_{\text{even}} \not\subseteq \mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ follows from Lemma 4.3.2. □

Claim 4.3.8. *There is a GJFA M such that $L(M_{\blacktriangleright\blacktriangleleft\curvearrowright}) = \{w \in \Sigma^* : w \text{ is an even palindrome}\}$.*

Proof. Consider an arbitrary alphabet Σ. Define the GJFA $M_{\blacktriangleright\blacktriangleleft\curvearrowright} = (\{f\}, \Sigma, R, f, \{f\})$ where $R = \{(f, a, f) : a \in \Sigma\}$. We follow Claim 4.3.5 and its proof, which shows that every $x \in L(M_{\blacktriangleright\blacktriangleleft\curvearrowright})$ can be written as $x = u_1u_2 \cdots u_nu_n \cdots u_2u_1$, where $n \geq 1$, and $u_i \in \Sigma^*$, $1 \leq i \leq n$. Observe that we use only rules reading single symbols; thus we can even say that $u_i \in (\Sigma \cup \{\varepsilon\})$, $1 \leq i \leq n$, which, in fact, models the string pattern of even palindromes. Moreover, we use only one sole state that can accept all symbols from Σ, therefore, $L(M_{\blacktriangleright\blacktriangleleft\curvearrowright}) = \{w \in \Sigma^* : w \text{ is an even palindrome}\}$. □

Lemma 4.3.9. *For every SEL $K_{m,n}$, there is a GJFA M such that $K_{m,n} = L(M_{\blacktriangleright\blacktriangleleft\curvearrowright})$.*

Proof. Let m and n be positive integers. Consider any SEL over an alphabet Σ,

$$K_{m,n} = \bigcup_{h=1}^{m} \left\{ u_{h,1}u_{h,2} \cdots u_{h,n}v_h^i v_h^i u_{h,n} \cdots u_{h,2}u_{h,1} : i \geq 0,\ u_{h,k}, v_h \in \Sigma^*,\ 1 \leq k \leq n \right\}.$$

Define the GJFA $M_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q, \Sigma, R, \langle s \rangle, F)$, where Q, R, and F are constructed in the following way:
(1) Add $\langle s \rangle$ to Q.
(2) Add $\langle h, k \rangle$ to Q, for all $1 \leq h \leq m$, $1 \leq k \leq n+1$.
(3) Add $\langle h, n+1 \rangle$ to F, for all $1 \leq h \leq m$.
(4) Add $(\langle s \rangle, \varepsilon, \langle h, 1 \rangle)$ to R, for all $1 \leq h \leq m$.
(5) Add $(\langle h, k \rangle, u_{h,k}, \langle h, k+1 \rangle)$ to R, for all $1 \leq h \leq m$, $1 \leq k \leq n$.
(6) Add $(\langle h, n+1 \rangle, v_h, \langle h, n+1 \rangle)$ to R, for all $1 \leq h \leq m$.

We follow Claim 4.3.5 and its proof. Observe that M starts from $\langle s \rangle$ by jumping to an arbitrary state $\langle h, 1 \rangle$, where $1 \leq h \leq m$. Then, the first jump consecutively reads $u_{h,1}u_{h,2} \cdots u_{h,n}$, and the second jump consecutively reads $u_{h,n} \cdots u_{h,2}u_{h,1}$, until M ends up in the final state $\langle h, n+1 \rangle$. Here, both jumps can arbitrarily many times read v_h. As a result, M accepts $u_{h,1}u_{h,2} \cdots u_{h,n}v_h^i v_h^i u_{h,n} \cdots u_{h,2}u_{h,1}$, for all $1 \leq h \leq m$, where $i \geq 0$, $u_{h,k}, v_h \in \Sigma^*$, $1 \leq k \leq n$; therefore, $K_{m,n} = L(M_{\blacktriangleright\blacktriangleleft\curvearrowright})$. □

Lemma 4.3.10. *For every SEL $K_{m,n}$, there is a right-linear grammar G such that $K_{m,n} = L(G)$.*

Proof. Let m and n be positive integers. Consider any SEL over an alphabet Σ,

$$K_{m,n} = \bigcup_{h=1}^{m} \left\{ u_{h,1}u_{h,2} \cdots u_{h,n}v_h^i v_h^i u_{h,n} \cdots u_{h,2}u_{h,1} : i \geq 0,\ u_{h,k}, v_h \in \Sigma^*,\ 1 \leq k \leq n \right\}.$$

Define the right-linear grammar $G = (N, \Sigma, P, \langle s \rangle)$, where N and P are constructed in the following way:
(1) Add $\langle s \rangle$ to N.
(2) Add $\langle h, 1 \rangle$ and $\langle h, 2 \rangle$ to N, for all $1 \leq h \leq m$.
(3) Add $\langle s \rangle \to \langle h, 1 \rangle$ to P, for all $1 \leq h \leq m$.
(4) Add $\langle h, 1 \rangle \to u_{h,1}u_{h,2} \cdots u_{h,n}\langle h, 2 \rangle$ to P, for all $1 \leq h \leq m$.
(5) Add $\langle h, 2 \rangle \to v_n v_n \langle h, 2 \rangle$ to P, for all $1 \leq h \leq m$.
(6) Add $\langle h, 2 \rangle \to u_{h,n} \cdots u_{h,2}u_{h,1}$ to P, for all $1 \leq h \leq m$.

Observe that, at the beginning, G has to change the nonterminal $\langle s \rangle$ to an arbitrary nonterminal $\langle h, 1 \rangle$, where $1 \leq h \leq m$. Then, it generates $u_{h,1}u_{h,2} \cdots u_{h,n}$ and the nonterminal $\langle h, 2 \rangle$. Here, it can arbitrarily many times generate $v_n v_n$ and ultimately finish the generation with $u_{h,n} \cdots u_{h,2}u_{h,1}$. As a result, G generates $u_{h,1}u_{h,2} \cdots u_{h,n}(v_h v_h)^i u_{h,n} \cdots u_{h,2}u_{h,1}$, for all $1 \leq h \leq m$, where $i \geq 0$, $u_{h,k}, v_h \in \Sigma^*$, $1 \leq k \leq n$, which is indistinguishable from $u_{h,1}u_{h,2} \cdots u_{h,n}v_h^i v_h^i u_{h,n} \cdots u_{h,2}u_{h,1}$; therefore, $K_{m,n} = L(G)$. □

Theorem 4.3.11. ***SEL*** $\subset$ ***REG***$_{even}$.

Proof. **SEL** $\subseteq$ **REG**$_{\text{even}}$ follows from Lemma 4.3.10 and the structure of its proof. **REG**$_{\text{even}}$ $\not\subseteq$ **SEL** follows from Lemma 4.3.9 and Lemma 4.3.2. □

Theorem 4.3.12. ***SEL*** $\subset \mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$.

Proof. **SEL** $\subseteq \mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ follows from Lemma 4.3.9. $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright} \not\subseteq$ **SEL** follows from Theorem 4.3.11 and Claim 4.3.8 because a subfamily of the family of regular languages surely cannot contain a non-trivial language of all even palindromes. □

Theorem 4.3.13. *The following pairs of language families are incomparable:*
(i) $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ *and* ***REG*** *(****REG****$_{even}$*)*;*
(ii) $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ *and* ***FIN*** *(****FIN****$_{even}$*)*.*

Proof. $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright} \not\subseteq$ **REG** (**REG**$_{\text{even}}$) and $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright} \not\subseteq$ **FIN** (**FIN**$_{\text{even}}$) follow from Claim 4.3.8, Theorem 4.3.11, and Theorem 4.3.12 (and Lemma 4.3.2). **REG** (**REG**$_{\text{even}}$) $\not\subseteq \mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ and **FIN** (**FIN**$_{\text{even}}$) $\not\subseteq \mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ follow from Lemma 4.3.2. Moreover, $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ clearly contains the regular language $\{a^{2n} : n \geq 0\}$ and finite language $\{aa\}$. □

Open Problem 4.3.14. $(\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright} -$ ***SEL*** $) \cap$ ***REG*** $= \emptyset$*?*

4.3.3 Left-Right 2-Jumping Relation

Claim 4.3.15. *Let $M = (Q, \Sigma, R, s, F)$ be a GJFA; then, every $x \in L(M_{\blacktriangleleft\blacktriangleright\curvearrowright})$ can be written as $x = u_n \cdots u_2 u_1 u_1 u_2 \cdots u_n$, where $n \geq 1$, and $u_i \in \Sigma^*$, $1 \leq i \leq n$.*

Proof. Consider any GJFA $M_{\blacktriangleleft\blacktriangleright\curvearrowright} = (Q, \Sigma, R, s, F)$. Since we work with the left-right 2-jumps, the first jump can move only to the left, and the second jump can move only to the right. Observe that if the configuration of M is of the form $upvpw$, where $u, v, w \in \Sigma^*$, and $p \in Q$, then M cannot read the symbols in v anymore. Also, observe that this covers the situation when M starts to accept $x \in \Sigma^*$ from any other configuration than $yssz$, where $y, z \in \Sigma^*$ such that $x = yz$. Therefore, to read the whole input string, M has to start in the configuration $yssz$, and it cannot jump over any symbols during the whole process. Consequently, since both jumps follow the same rule, they have to read the same corresponding strings and ultimately finish at the ends of the input string. Therefore, every $x \in L(M_{\blacktriangleleft\blacktriangleright\curvearrowright})$ can be written as $x = u_n \cdots u_2 u_1 u_1 u_2 \cdots u_n$, where $n \geq 1$, and $u_i \in \Sigma^*$, $1 \leq i \leq n$. □

Lemma 4.3.16. *For every GJFA M, there is a GJFA N such that $L(M_{\blacktriangleleft\blacktriangleright\curvearrowright}) = L(N_{\blacktriangleright\blacktriangleleft\curvearrowright})$.*

Proof. Consider any *GJFA* $M_{\blacktriangleleft\blacktriangleright\curvearrowright} = (Q, \Sigma, R_1, s_1, F)$. Without loss of generality, assume that $s_2 \notin Q$. Define the GJFA $N_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q \cup \{s_2\}, \Sigma, R_2, s_2, \{s_1\})$, where R_2 is constructed in the following way:
(1) For each $(p, y, q) \in R_1$, add (q, y, p) to R_2.
(2) For each $f \in F$, add (s_2, ε, f) to R_2.

Note that this construction resembles the well-known conversion technique for finite automata which creates a finite automaton that accepts the reversal of the original language. However, in this case, the effect is quite different. We follow Claims 4.3.5 and 4.3.15. Consider any $x \in L(M_{\blacktriangleleft\blacktriangleright\curvearrowright})$. We can surely find $x = u_n \cdots u_2 u_1 u_1 u_2 \cdots u_n$, where $n \geq 1$, and $u_i \in \Sigma^*$, $1 \leq i \leq n$, such that N reads $u_n \cdots u_2 u_1$ and $u_1 u_2 \cdots u_n$ in the reverse order. Moreover, in N, both jumps have their direction reversed, compared to jumps in M, and thus they start on the opposite ends of their parts, which is demonstrated in the mentioned claims. Consequently, if each jump in N reads its part reversely and from the opposite end, then N reads the same $u_n \cdots u_2 u_1 u_1 u_2 \cdots u_n$ as M. Finally, N surely cannot accept anything new that is not accepted by M. Thus, $L(M_{\blacktriangleleft\blacktriangleright\curvearrowright}) = L(N_{\blacktriangleright\blacktriangleleft\curvearrowright})$. □

Lemma 4.3.17. *For every GJFA M, there is a GJFA N such that $L(M_{\blacktriangleright\blacktriangleleft\curvearrowright}) = L(N_{\blacktriangleleft\blacktriangleright\curvearrowright})$.*

Proof. The construction and reasoning is exactly the same as in Lemma 4.3.16. □

Theorem 4.3.18. $\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright} = \mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$.

Proof. $\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright} \subseteq \mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ follows from Lemma 4.3.16. $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright} \subseteq \mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$ follows from Lemma 4.3.17. □

Other properties of this language family thus coincide with Section 4.3.2.

The results concerning the accepting power of GJFAs that perform right-left and left-right 2-jumps are summarized in Figure 4.1.

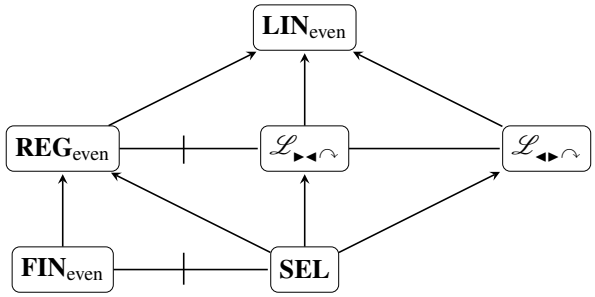

Figure 4.1: A hierarchy of language families closely related to the right-left and left-right 2-jumps is shown. If there is a line or an arrow from family X to family Y in the figure, then $X = Y$ or $X \subset Y$, respectively. A crossed line represents the incomparability between connected families.

4.3.4 Right-Right 2-Jumping Relation

Example 4.3.19. Consider the GJFA

$$M_{\blacktriangleright\blacktriangleright\curvearrowright} = (\{s, p, f\}, \Sigma, R, s, \{f\}),$$

where $\Sigma = \{a, b, c\}$ and R consists of the rules (s, ab, p) and (p, c, f). Starting from s, M has to read two times ab and two times c. Observe that if the first jump skips (jumps over) some symbols, then they cannot be ever read afterwards. However, the second jump is not so harshly restricted and can potentially skip some symbols which will be read later by the first jump. Therefore, the accepted language is

$$L(M_{\blacktriangleright\blacktriangleright\curvearrowright}) = \{ababcc, abcabc\}.$$

Example 4.3.20. Consider the GJFA

$$M_{\blacktriangleright\blacktriangleright\curvearrowright} = (\{s, f\}, \Sigma, R, s, \{f\}),$$

where $\Sigma = \{a, b\}$ and R consists of the rules (s, b, f) and (f, a, f). Starting from s, M has to read two times b, and then it can arbitrarily many times read two times a. Both jumps behave the same way as in Example 4.3.19. Observe that when we consider no skipping of symbols, then M reads ba^nba^n, $n \geq 0$. Nevertheless, when we consider the skipping with

the second jump, then the second b can also occur arbitrarily closer to the first b; until they are neighbors, and M reads bba^{2n}, $n \geq 0$. When combined together, the result is

$$L(M_{\blacktriangleright\blacktriangleright\curvearrowright}) = \{ba^n ba^n a^{2m} : n, m \geq 0\}.$$

Observe that this is clearly a non-regular context-free language.

Example 4.3.21. Consider the GJFA

$$M_{\blacktriangleright\blacktriangleright\curvearrowright} = (\{s, f\}, \Sigma, R, s, \{f\}),$$

where $\Sigma = \{a, b, c, d\}$ and $R = \{(s, y, f) : y \in \Sigma\} \cup \{(f, y, f) : y \in \Sigma\}$. Starting from s, M has to read two times some symbol from Σ, and then it can arbitrarily many times read two times any symbols from Σ. Again, both jumps behave the same way as in Example 4.3.19. Consider the special case when the second jump consistently jumps over one symbol each time (except the last step) during the whole process. In such a case, the accepted strings can be written as $u_1 u'_1 u_2 u'_2 \cdots u_n u'_n$, where $n \geq 1$, $u_i, u'_i \in \Sigma$, $u_i = u'_i$, $1 \leq i \leq n$. Observe that the symbols without primes are read by the first jump, and the symbols with primes are read by the second jump. Moreover, such strings can be surely generated by a right-linear grammar. Nevertheless, now consider no special case. Observe that, in the accepted strings, the symbols with primes can be arbitrarily shifted to the right over symbols without primes. This creates a more complex structure, due to $u_i = u'_i$, with multiple crossed agreements. Lastly, consider the other border case with no skipping of any symbols at all. Then, the accepted strings can be written as ww, where $w \in \Sigma^+$. Such strings represent the reduplication phenomenon—the well-known example of non-context-free languages (see Chapter 3.1 in [78]). As a result, due to the unbound number of crossed agreements, we can safely state that $L(M_{\blacktriangleright\blacktriangleright\curvearrowright})$ is a non-context-free language.

This statement can be formally proven by contradiction. Assume that $L(M_{\blacktriangleright\blacktriangleright\curvearrowright})$ is a context-free language. The family of context-free languages is closed under intersection with regular sets. Let $K = L(M_{\blacktriangleright\blacktriangleright\curvearrowright}) \cap ab^+c^+dab^+c^+d$. Consider the previous description. Observe that this selects strings where $u_1 = a$ and $u'_n = d$. Since there are only exactly two symbols a and two symbols d in each selected string, we know where precisely both jumps start and end. And since the second jump starts after the position where the first jump ends, we also know that this, in fact, follows the special border case of behavior with no skipping of any symbols at all. Consequently, $K = \{ab^n c^m dab^n c^m d : n, m \geq 1\}$. However, K is clearly a non-context-free language (see Chapter 3.1 in [78])—a contradiction with the assumption that $L(M_{\blacktriangleright\blacktriangleright\curvearrowright})$ is a context-free language. Therefore, $L(M_{\blacktriangleright\blacktriangleright\curvearrowright})$ is a non-context-free language.

Theorem 4.3.22. $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright} \subset \boldsymbol{CS}_{even}$.

Proof. Clearly, any GJFA $M_{\blacktriangleright\blacktriangleright\curvearrowright}$ can be simulated by linear bounded automata; thus $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright} \subseteq \mathbf{CS}$. Due to Lemma 4.3.2, we can safely exclude all languages containing odd-length strings. $\mathbf{CS}_{\text{even}} \not\subseteq \mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ also follows from Lemma 4.3.2. □

Lemma 4.3.23. *Let $n \geq 0$, and let M be any GJFA. Furthermore, let every $x \in L(M_{\blacktriangleright\blacktriangleright\curvearrowright})$ satisfy either $|x| \leq n$ or $\text{alph}(x) = 1$. Then, there exists a right-linear grammar G such that $L(M_{\blacktriangleright\blacktriangleright\curvearrowright}) = L(G)$.*

Proof. Let $n \geq 0$. Consider any GJFA $M_{\blacktriangleright\blacktriangleright\curvearrowright}$ where every $x \in L(M_{\blacktriangleright\blacktriangleright\curvearrowright})$ satisfies either $|x| \leq n$ or $\text{alph}(x) = 1$. Define the right-linear grammar G in the following way: Observe

that the number of x for which holds $|x| \leq n$ must be finite, therefore, for each such x, we can create a separate rule that generates x in G. On the other hand, the number of x for which holds $\text{alph}(x) = 1$ can be infinite, however, every such x is defined by a finite number of rules in M. And we can surely convert these rules (p, y, q) from M into rules in G in such a way that they generate y^2 and simulate the state transitions of M. Consequently, since here the positions of symbols are ultimately irrelevant, these rules properly simulate the results of 2-jumps in M. Therefore, $L(M_{\blacktriangleright\blacktriangleright\curvearrowright}) = L(G)$. □

Theorem 4.3.24. *The following pairs of language families are incomparable:*
(i) $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$ *and* ***CF*** *(****CF****$_{even}$);*
(ii) $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$ *and* ***REG*** *(****REG****$_{even}$);*
(iii) $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$ *and* ***FIN*** *(****FIN****$_{even}$).*

Proof. $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright} \not\subseteq$ **CF** (**CF**$_{\text{even}}$), $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright} \not\subseteq$ **REG** (**REG**$_{\text{even}}$), and $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright} \not\subseteq$ **FIN** (**FIN**$_{\text{even}}$) follow from Example 4.3.21. **CF** (**CF**$_{\text{even}}$) $\not\subseteq \mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, **REG** (**REG**$_{\text{even}}$) $\not\subseteq \mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and **FIN** (**FIN**$_{\text{even}}$) $\not\subseteq \mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$ follow from Lemma 4.3.2. Moreover, observe that $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$ clearly contains the context-free language from Example 4.3.20, regular language $\{a^{2n} : n \geq 0\}$, and finite language from Example 4.3.19. □

4.3.5 Left-Left 2-Jumping Relation

Example 4.3.25. Consider the GJFA

$$M_{\blacktriangleleft\blacktriangleleft\curvearrowright} = (\{s, p, f\}, \Sigma, R, s, \{f\}),$$

where $\Sigma = \{a, b, c\}$ and R consists of the rules (s, c, p) and (p, ab, f). Starting from s, M has to read two times c and two times ab. Observe that if the second jump skips some symbols, then they cannot be ever read afterwards. However, the first jump is not so harshly restricted and can potentially skip some symbols which will be read later by the second jump. Note that this precisely resembles the inverted behavior of the right-right 2-jumping relation. As a result, the language is

$$L(M_{\blacktriangleleft\blacktriangleleft\curvearrowright}) = \{ababcc, abacbc, abcabc\}.$$

Example 4.3.26. Consider the GJFA

$$M_{\blacktriangleleft\blacktriangleleft\curvearrowright} = (\{s, f\}, \Sigma, R, s, \{f\}),$$

where $\Sigma = \{a, b\}$ and R consists of the rules (s, a, s) and (s, b, f). Starting from s, M can arbitrarily many times read two times a, and, as the last step, it has to read two times b. Both jumps behave the same way as in Example 4.3.25. Observe that when we consider no skipping of symbols, then M reads ba^nba^n, $n \geq 0$. Nevertheless, when we consider the skipping with the first jump, then the second b can also occur arbitrarily closer to the first b, since the first jump can now read symbols a also behind this second b. Consequently, the accepted language is

$$L(M_{\blacktriangleleft\blacktriangleleft\curvearrowright}) = \{ba^nba^na^{2m} : n, m \geq 0\}.$$

Note that this is the same language as in Example 4.3.20.

Example 4.3.27. Consider the GJFA

$$M_{\blacktriangleleft\blacktriangleleft\curvearrowright} = (\{s, f\}, \Sigma, R, s, \{f\}),$$

where $\Sigma = \{a, b, c, d\}$ and $R = \{(s, y, f) : y \in \Sigma\} \cup \{(f, y, f) : y \in \Sigma\}$. Starting from s, M has to read two times some symbol from Σ, and then it can arbitrarily many times read two times any symbols from Σ. Both jumps behave the same way as in Example 4.3.25, and the overall behavior tightly follows Example 4.3.21. In the special case where the first jump consistently jumps over one symbol each time (except the last step) during the whole process, the accepted strings can be written as $u'_n u_n \cdots u'_2 u_2 u'_1 u_1$, where $n \geq 1$, $u'_i, u_i \in \Sigma$, $u'_i = u_i$, $1 \leq i \leq n$. The symbols with primes are read by the first jump, and the symbols without primes are read by the second jump. With no special case, the symbols with primes can be arbitrarily shifted to the left over the symbols without primes, which creates a more complex structure with multiple crossed agreements and ultimately also the structure of the reduplication phenomenon. As a result, we can safely state that $L(M_{\blacktriangleleft\blacktriangleleft\curvearrowright})$ is a non-context-free language, and this statement can be formally proven in the same way as in Example 4.3.21.

Theorem 4.3.28. $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright} \subset \boldsymbol{CS}_{even}$.

Proof. The reasoning is identical to Theorem 4.3.22. □

Lemma 4.3.29. *Let $n \geq 0$, and let M be any GJFA. Furthermore, let every $x \in L(M_{\blacktriangleleft\blacktriangleleft\curvearrowright})$ satisfy either $|x| \leq n$ or $\mathrm{alph}(x) = 1$. Then, there exists a right-linear grammar G such that $L(M_{\blacktriangleleft\blacktriangleleft\curvearrowright}) = L(G)$.*

Proof. The reasoning is exactly the same as in Lemma 4.3.23. □

Theorem 4.3.30. *The following pairs of language families are incomparable:*
(i) $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ and $\boldsymbol{CF}$ ($\boldsymbol{CF}_{even}$);
(ii) $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ and $\boldsymbol{REG}$ ($\boldsymbol{REG}_{even}$);
(iii) $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ and $\boldsymbol{FIN}$ ($\boldsymbol{FIN}_{even}$).

Proof. $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright} \not\subseteq \mathbf{CF}$ ($\mathbf{CF}_{\text{even}}$), $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright} \not\subseteq \mathbf{REG}$ ($\mathbf{REG}_{\text{even}}$), and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright} \not\subseteq \mathbf{FIN}$ ($\mathbf{FIN}_{\text{even}}$) follow from Example 4.3.27. $\mathbf{CF}$ ($\mathbf{CF}_{\text{even}}$) $\not\subseteq \mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$, $\mathbf{REG}$ ($\mathbf{REG}_{\text{even}}$) $\not\subseteq \mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$, and $\mathbf{FIN}$ ($\mathbf{FIN}_{\text{even}}$) $\not\subseteq \mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ follow from Lemma 4.3.2. Moreover, $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ contains the context-free language from Example 4.3.26, regular language $\{a^{2n} : n \geq 0\}$, and finite language from Example 4.3.25. □

Claim 4.3.31. *There is no GJFA $M_{\blacktriangleright\blacktriangleright\curvearrowright}$ that accepts $\{ababcc, abacbc, abcabc\}$.*

Proof. By contradiction. Let $K = \{ababcc, abacbc, abcabc\}$. Assume that there is a GJFA M such that $L(M_{\blacktriangleright\blacktriangleright\curvearrowright}) = K$. Observe that each string in K contains three pairs of symbols, therefore, to effectively read such a string, we need a maximum of three chained rules in M or less. (Note that additional rules reading ε do not affect results.) Moreover, due to the nature of strings in K, we need to consider only such chains of rules where, in the result, a precedes b, and b precedes c. Therefore, we can easily try all possibilities and calculate their resulting sets. Surely, $L(M_{\blacktriangleright\blacktriangleright\curvearrowright})$ must be a union of some of these sets:
(i) if M reads abc, the set is $\{abcabc\}$;
(ii) if M reads ab, and c, the set is $\{ababcc, abcabc\}$;
(iii) if M reads a, and bc, the set is $\{aabcbc, abacbc, abcabc\}$;

(iv) if M reads a, b, and c, the set is $\{aabbcc, ababcc, aabcbc, abacbc, abcabc\}$.

Clearly, no union of these sets can result in K—a contradiction with the assumption that $L(M_{\blacktriangleright\blacktriangleright\curvearrowright}) = K$ exists. Thus, there is no GJFA $M_{\blacktriangleright\blacktriangleright\curvearrowright}$ that accepts $\{ababcc, abacbc, abcabc\}$. □

Claim 4.3.32. *There is no GJFA $M_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ that accepts $\{ababcc, abcabc\}$.*

Proof. By contradiction. Let $K = \{ababcc, abcabc\}$. Assume that there is a GJFA M such that $L(M_{\blacktriangleleft\blacktriangleleft\curvearrowright}) = K$. By the same reasoning as in the proof of Claim 4.3.31, $L(M_{\blacktriangleleft\blacktriangleleft\curvearrowright})$ must be a union of some of these sets:

(i) if M reads abc, the set is $\{abcabc\}$;
(ii) if M reads c, and ab, the set is $\{ababcc, abacbc, abcabc\}$;
(iii) if M reads bc, and a, the set is $\{aabcbc, abcabc\}$;
(iv) if M reads c, b, and a, the set is $\{aabbcc, aabcbc, ababcc, abacbc, abcabc\}$.

Clearly, no union of these sets can result in K. Thus, there is no GJFA $M_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ that accepts $\{ababcc, abcabc\}$. □

Theorem 4.3.33. *$\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$ and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ are incomparable.*

Proof. $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright} \not\subseteq \mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ follows from Example 4.3.19 and Claim 4.3.32. $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright} \not\subseteq \mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$ follows from Example 4.3.25 and Claim 4.3.31. Moreover, both $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$ and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ clearly contain the same language from Examples 4.3.20 and 4.3.26. □

The results concerning the accepting power of GJFAs that perform right-right and left-left 2-jumps are summarized in Figure 4.2.

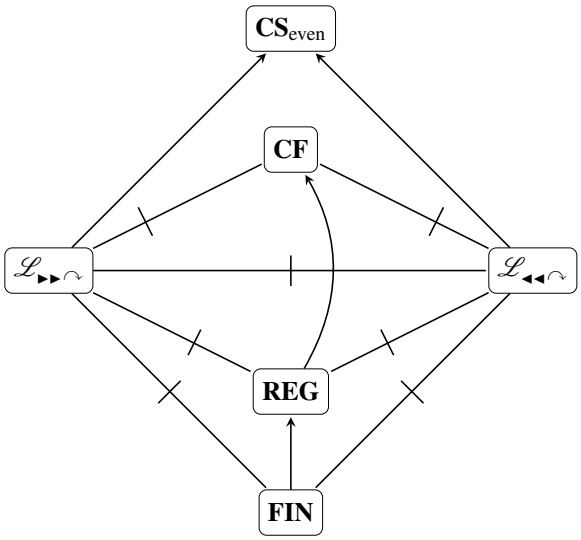

Figure 4.2: A hierarchy of language families closely related to the right-right and left-left 2-jumps is shown. If there is a line or an arrow from family X to family Y in the figure, then $X = Y$ or $X \subset Y$, respectively. A crossed line represents the incomparability between connected families.

4.4 Closure Properties

In this section, we show the closure properties of $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$, $\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$, $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ under various operations. Recall that, by Theorem 4.3.18, $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ and $\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$ are equivalent, and so their closure properties coincide.

Theorem 4.4.1. *All $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ ($\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$), $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ are not closed under endmarking.*

Proof. This result directly follows from Lemma 4.3.2—the inability to read an odd number of symbols from the input string. □

Theorem 4.4.2. *$\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ ($\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$) is closed under endmarking on both sides.*

Proof. Consider any GJFA $M_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q, \Sigma, R, s, F)$. Without loss of generality, assume that $s' \notin Q$ and $\# \notin \Sigma$. Define GJFA $N_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q \cup \{s'\}, \Sigma \cup \{\#\}, R \cup \{(s', \#, s)\}, s', F)$. Then, by Claim 4.3.5, every $x \in L(N_{\blacktriangleright\blacktriangleleft\curvearrowright})$ can be surely written as $x = \#u_2u_3 \cdots u_nu_n \cdots u_3u_2\#$, where $n \geq 2$, and $u_i \in \Sigma^*$, $2 \leq i \leq n$. □

Theorem 4.4.3. *Both $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$ and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ are not closed under endmarking on both sides.*

Proof. Since both jumps always read the same strings in the same direction, they clearly cannot reliably define the endmarking on the opposite sides of the input string in the general case. □

Theorem 4.4.4. *All $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ ($\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$), $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ are not closed under concatenation.*

Proof. This can be easily proven by contradiction. Consider two simple languages $\{aa\}$ and $\{bb\}$, which clearly belong into $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$, $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$. Assume that $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$, $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ are closed under concatenation. Therefore, the resulting language $\{aabb\}$ also have to belong into $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$, $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$. However, such a language does not satisfy the string form for $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ from Claim 4.3.5, and there is no GJFA $M_{\blacktriangleright\blacktriangleright\curvearrowright}$ or GJFA $N_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ that can define such a language. Observe that M and N cannot accept $aabb$ with a single 2-jump, and that the rules for multiple 2-jumps define broader languages, e.g. $\{abab, aabb\}$. □

Theorem 4.4.5. *All $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ ($\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$), $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ are not closed under square.*

Proof. Consider the language $L = \{aa, bb\}$, which clearly belongs into $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$, $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$. Therefore, $L^2 = \{aaaa, aabb, bbaa, bbbb\}$ should also belong into these language families. However, observe the string $aabb$, it causes the same problems as in the proof of Theorem 4.4.4. This string does not satisfy the string form for $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ from Claim 4.3.5. Moreover, there is no GJFA $M_{\blacktriangleright\blacktriangleright\curvearrowright}$ or GJFA $N_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ that can simultaneously accept the required string $aabb$ and reject the unwanted string $abab$. □

Theorem 4.4.6. *All $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ ($\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$), $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ are not closed under shuffle.*

Proof. Consider two simple languages $\{aa\}$ and $\{bb\}$, which clearly belong into $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$, $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$. Therefore, the resulting language of their shuffle $\{aabb, abab, baab, abba, baba, bbaa\}$ should also belong into these language families. However, several strings from this language do not satisfy the string form for $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ from Claim 4.3.5. Moreover, there is surely no GJFA $M_{\blacktriangleright\blacktriangleright\curvearrowright}$ or GJFA $N_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ that can accept the string $baab$ or $abba$, since these strings do not contain two identical sequences of symbols that could be properly synchronously read. □

Theorem 4.4.7. *All $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ ($\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$), $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ are closed under union.*

Proof. Let o be one of the relations $\blacktriangleright\blacktriangleleft\curvearrowright$, $\blacktriangleright\blacktriangleright\curvearrowright$, and $\blacktriangleleft\blacktriangleleft\curvearrowright$; and $M_o = (Q_1, \Sigma_1, R_1, s_1, F_1)$, and $N_o = (Q_2, \Sigma_2, R_2, s_2, F_2)$ be two GJFAs. Without loss of generality, assume that $Q_1 \cap Q_2 = \emptyset$ and $s \notin (Q_1 \cup Q_2)$. Define the GJFA

$$H_o = (Q_1 \cup Q_2 \cup \{s\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{(s, \varepsilon, s_1), (s, \varepsilon, s_2)\}, s, F_1 \cup F_2).$$

Observe that $L(H_o) = L(M_o) \cup L(N_o)$ holds in all modes. Indeed, the leading 2-jump only selects whether H_o enters M_o or N_o, and this leading 2-jump introduces no other new configuration to the configurations of M_o and N_o. □

Theorem 4.4.8. *All $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ ($\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$), $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ are not closed under complement.*

Proof. Consider Lemma 4.3.2—that all 2-jumping modes can only accept even-length input strings. As a result, every complement has to contain at least all odd-length strings, and thus it cannot be defined by any 2-jumping mode. □

Theorem 4.4.9. *$\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ ($\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$) is closed under intersection with regular languages.*

Proof. Consider any GJFA $M_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q_1, \Sigma, R_1, s_1, F_1)$ and FA $N = (Q_2, \Sigma, R_2, s_2, F_2)$. We can define a new GJFA $H_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q_3, \Sigma, R_3, s_3, F_3)$ that simulates both M and N in the same time and that accepts the input string x if and only if both M and N also accept x. Note that the requirement of identical Σ does not affect the generality of the result. We are going to use two auxiliary functions that will help us with the construction of H. First, $\mathrm{Fw}(N, p, str)$ that accepts three parameters: N which is some FA, p which is some state of N, and str which is some string. This function returns the set of states in which N can end up if N is in the state p and reads str. Second, $\mathrm{Bw}(N, p, str)$ that also accepts the same parameters: N which is some FA, p which is some state of N, and str which is some string. This function returns the set of states from which N reads str and ends in the state p. We are not giving full details of these functions here since they only incorporate the well-known standard techniques for finite automata. With this, we construct Q_3, R_3, and F_3 in the following way:

(1) Add s_3 to Q_3.
(2) Add $\langle p, q, r\rangle$ to Q_3, for all $(p, q, r) \in Q_1 \times Q_2 \times Q_2$.
(3) Add $\langle p, q, q\rangle$ to F_3, for all $(p, q) \in F_1 \times Q_2$.
(4) Add $(s_3, \varepsilon, \langle s_1, s_2, f\rangle)$ to R_3, for all $f \in F_2$.
(5) For each $(p, a, q) \in R_1$ and $r_1, t_1 \in Q_2$, add $(\langle p, r_1, t_1\rangle, a, \langle q, r_2, t_2\rangle)$ to R_3, for all $(r_2, t_2) \in \mathrm{Fw}(N, r_1, a) \times \mathrm{Bw}(N, t_1, a)$.

Observe that H handles three distinct things in its states $\langle p, q, r\rangle$: p represents the original state of M, q simulates the first part of N in the classical forward way, and r simulates the second part of N in the backward way. At the beginning, H makes a 2-jump from the initial state s_3 into one of the states $\langle s_1, s_2, f\rangle$, where $f \in F_2$, and the main part of the simulation starts. In each following step, H can only make a 2-jump if the similar 2-jump is also in M and if N can read the same string as M from both opposite sides with the current states. This part ends when there are no valid 2-jumps or when H reads the whole input string. If H processes the whole input string, we can recognize the valid final state $\langle p, q, r\rangle$ in the following way: p has to be the original final state of M, and q must be the same as r so that the simulation of N from two opposite sides can be connected in the middle. As a result, $L(H_{\blacktriangleright\blacktriangleleft\curvearrowright}) = L(M_{\blacktriangleright\blacktriangleleft\curvearrowright}) \cap L(N)$. □

Theorem 4.4.10. *$\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ ($\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$) is closed under intersection.*

Proof. Consider any GJFA $M_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q_1, \Sigma, R_1, s_1, F_1)$ and GJFA $N_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q_2, \Sigma, R_2, s_2, F_2)$. We can define a new GJFA $H_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q, \Sigma, R, s, F)$ that simulates both M and N in the same time such that $L(H_{\blacktriangleright\blacktriangleleft\curvearrowright}) = L(M_{\blacktriangleright\blacktriangleleft\curvearrowright}) \cap L(N_{\blacktriangleright\blacktriangleleft\curvearrowright})$. To support the construction of Q and R, define $\Sigma^{\leq h} = \bigcup_{i=0}^{h} \Sigma^i$, and let k be the maximum length of the right-hand sides of the rules from $R_1 \cup R_2$. First, set Q to $\{\langle q_1, x, x', q_2, y, y' \rangle : q_1 \in Q_1,\ q_2 \in Q_2,\ x, x', y, y' \in \Sigma^{\leq 2k-1}\}$, F to $\{\langle f_1, \varepsilon, \varepsilon, f_2, \varepsilon, \varepsilon \rangle : f_1 \in F_1,\ f_2 \in F_2\}$, and $s = \langle s_1, \varepsilon, \varepsilon, s_2, \varepsilon, \varepsilon \rangle$. Then, we construct R in the following way:

(I) Add $(\langle p, x, x', q, y, y' \rangle, a, \langle p, xa, ax', q, ya, ay' \rangle)$ to R, for all $a \in \Sigma^{\leq k}$, $p \in Q_1$, $q \in Q_2$, and $x, x', y, y' \in \Sigma^{\leq 2k-1-|a|}$.

(II) For each $(p, a, p') \in R_1$, add $(\langle p, ax, x'a, q, y, y' \rangle, \varepsilon, \langle p', x, x', q, y, y' \rangle)$ to R, for all $x, x' \in \Sigma^{\leq 2k-1-|a|}$, $q \in Q_2$, and $y, y' \in \Sigma^{\leq 2k-1}$.

(III) For each $(q, b, q') \in R_2$, add $(\langle p, x, x', q, by, y'b \rangle, \varepsilon, \langle p, x, x', q', y, y' \rangle)$ to R, for all $p \in Q_1$, $x, x' \in \Sigma^{\leq 2k-1}$, and $y, y' \in \Sigma^{\leq 2k-1-|b|}$.

Observe that H stores six pieces of information in its compound states: (1) the state of M, (2) the buffered string (so called buffer) with up to $2k-1$ symbols read from the beginning of the input string to simulate the work of M on it, (3) the buffered string with up to $2k-1$ symbols read from the end of the input string to simulate the work of M on it, and the pieces (4), (5), and (6) are analogous to (1), (2), and (3) but for N, respectively.

Next, by the same reasoning as in the proof of Claim 4.3.5, we can assume that M and N start from the configurations $s_1 w s_1$ and $s_2 w s_2$, respectively, and neither of them can jump over any symbols during the reading. Using these assumptions, H simulates the work of M and N as follows. First, it reads by the rules from (I) a part of the input string and stores it in the buffers. Then, by the rules from (II) and (III), H processes the symbols from the buffers by the simulation of the rules from M and N. Whenever needed, H reads from the input string some additional symbols using the rules from (I). The input string is accepted by H if and only if the whole input string is read, all buffers are processed and emptied, and both (1) and (4) are the final states of M and N, respectively.

To justify the maximum size of the buffers in (2), (3), (5), and (6), consider the situation when the simulation of M needs to read the input string by the words of length k, but the right-hand sides of the simulated rules of N alternate between 1 and k symbols. Then, we can observe a situation when a buffer contains $k-1$ symbols and we have to read k additional symbols from the input string before we can process the first (or the last) k symbols of the buffer. The question remains, however, whether we can reliably exclude some of these situations and further decrease the size of the buffers in the states of H.

The rigorous proof of $L(H_{\blacktriangleright\blacktriangleleft\curvearrowright}) = L(M_{\blacktriangleright\blacktriangleleft\curvearrowright}) \cap L(N_{\blacktriangleright\blacktriangleleft\curvearrowright})$ is left to the reader. □

Theorem 4.4.11. *Both $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$ and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ are not closed under intersection and intersection with regular languages.*

Proof. Consider two GJFAs:

$$M_{\blacktriangleright\blacktriangleright\curvearrowright} = (\{s, r, p, f\}, \{a, b\}, \{(s, a, r), (r, bb, p), (p, a, f)\}, s, \{f\});$$
$$L(M_{\blacktriangleright\blacktriangleright\curvearrowright}) = \{abbaabba, abbabbaa, ababbba, ababbbaa, aabbabba, aabbbbaa\},$$
$$\text{and } N_{\blacktriangleright\blacktriangleright\curvearrowright} = (\{s, r, p, f\}, \{a, b\}, \{(s, a, r), (r, b, p), (p, ba, f)\}, s, \{f\});$$
$$L(N_{\blacktriangleright\blacktriangleright\curvearrowright}) = \{abbaabba, abbababa, ababbba, ababbaba, aabbabba, aabbbaba\}.$$

The intersection $L_\cap = L(M_{\blacktriangleright\blacktriangleright\curvearrowright}) \cap L(N_{\blacktriangleright\blacktriangleright\curvearrowright}) = \{abbaabba, ababbba, aabbabba\}$ should also belong into $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$. However, consider the simplest GJFA $P_{\blacktriangleright\blacktriangleright\curvearrowright}$ that can accept the string

aabbabba; it surely has to start with reading two times only one symbol a, then it can read two times bb together, and then it finishes by reading two times the symbol a. However, this is exactly the behavior of $M_{\blacktriangleright\blacktriangleright\curvearrowright}$, and we see that $L(M_{\blacktriangleright\blacktriangleright\curvearrowright})$ is a proper superset of $L_{\cap}$. Therefore, there cannot be any GJFA $H_{\blacktriangleright\blacktriangleright\curvearrowright}$ that defines $L_{\cap}$. Trivially, both $L(M_{\blacktriangleright\blacktriangleright\curvearrowright})$ and $L(N_{\blacktriangleright\blacktriangleright\curvearrowright})$ are also regular languages. The similar proof for $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ is left to the reader. □

Theorem 4.4.12. *$\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ ($\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$) is closed under mirror image.*

Proof. Consider any GJFA $M_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q, \Sigma, R_1, s, F)$. Define the GJFA $N_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q, \Sigma, R_2, s, F)$, where R_2 is constructed in the following way. For each $(p, a, q) \in R_1$, add $(p, \mathrm{mi}(a), q)$ to R_2. Note that, by Claim 4.3.5 and its proof, every $x \in L(M_{\blacktriangleright\blacktriangleleft\curvearrowright})$ can be written as $x = u_1 u_2 \cdots u_n u_n \cdots u_2 u_1$, where $n \geq 1$, and $u_i \in \Sigma^*$, $1 \leq i \leq n$; and where each u_i represents the string a from a certain rule. Observe that each x almost resembles an even palindrome. We just need to resolve the individual parts $|u_i| > 1$ for which the palindrome statement does not hold. Nevertheless, observe that if we simply mirror each u_i individually, it will create the mirror image of the whole x. As a result, $L(N_{\blacktriangleright\blacktriangleleft\curvearrowright})$ is a mirror image of $L(M_{\blacktriangleright\blacktriangleleft\curvearrowright})$. □

Theorem 4.4.13. *Both $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$ and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ are not closed under mirror image.*

Proof. Consider the language $K = \{ababcc, abcabc\}$, which is accepted by the GJFA

$$M_{\blacktriangleright\blacktriangleright\curvearrowright} = (\{s, r, f\}, \{a, b, c\}, \{(s, ab, r), (r, c, f)\}, s, \{f\}).$$

Therefore, the mirror language $K_{mi} = \{ccbaba, cbacba\}$ should also belong into $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$. However, consider the simplest GJFA $N_{\blacktriangleright\blacktriangleright\curvearrowright}$ that can accept the string *ccbaba*; it surely has to start with reading two times only symbol c, then it can read two times ba together. Even in such a case $L(N_{\blacktriangleright\blacktriangleright\curvearrowright}) = \{ccbaba, cbcaba, cbacba\}$; which is a proper superset of K_{mi}. Therefore, there cannot be any GJFA $H_{\blacktriangleright\blacktriangleright\curvearrowright}$ that defines K_{mi}. The similar proof for $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ is left to the reader. □

Theorem 4.4.14. *All $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ ($\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$), $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ are not closed under finite substitution.*

Proof. Consider the language $L = \{a^{2n} : n \geq 0\}$, which clearly belongs into $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$, $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$. Define the finite substitution $\varphi : \{a\}^* \to 2^{\{a\}^*}$ as $\varphi(a) = \{\varepsilon, a\}$. Observe that $\varphi(L)$ contains odd-length strings. However, in consequence of Lemma 4.3.2, we know that no 2-jumping mode can accept such strings. □

Theorem 4.4.15. *$\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ ($\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$) is closed under homomorphism and ε-free homomorphism.*

Proof. Consider any GJFA $M_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q, \Sigma, R_1, s, F)$ and an arbitrary homomorphism $\varphi : \Sigma^* \to \Delta^*$. Define the GJFA $N_{\blacktriangleright\blacktriangleleft\curvearrowright} = (Q, \Delta, R_2, s, F)$, where R_2 is constructed in the following way. For each $(p, a, q) \in R_1$, add $(p, \varphi(a), q)$ to R_2. Observe that by Claim 4.3.5 and its proof, every $x \in L(M_{\blacktriangleright\blacktriangleleft\curvearrowright})$ can be written as $x = u_1 u_2 \cdots u_n u_n \cdots u_2 u_1$, where $n \geq 1$, and $u_i \in \Sigma^*$, $1 \leq i \leq n$; and where each u_i represents the string a from a certain rule. Then, every $y \in L(N_{\blacktriangleright\blacktriangleleft\curvearrowright})$ can be surely written as $y = \varphi(u_1)\varphi(u_2)\cdots\varphi(u_n)\varphi(u_n)\cdots\varphi(u_2)\varphi(u_1)$, and clearly $\varphi(L(M_{\blacktriangleright\blacktriangleleft\curvearrowright})) = L(N_{\blacktriangleright\blacktriangleleft\curvearrowright})$. □

Theorem 4.4.16. *Both* $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$ *and* $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ *are not closed under homomorphism and* ε*-free homomorphism.*

Proof. Consider the language $K = \{abab, aabb\}$, which is accepted by the GJFA

$$M_{\blacktriangleright\blacktriangleright\curvearrowright} = (\{s, r, f\}, \{a, b\}, \{(s, a, r), (r, b, f)\}, s, \{f\}).$$

Define the ε-free homomorphism $\varphi : \{a, b\}^+ \to \{a, b, c\}^+$ as $\varphi(a) = a$ and $\varphi(b) = bc$. By applying φ to K, we get $\varphi(K) = \{abcabc, aabcbc\}$. Consider the simplest GJFA $N_{\blacktriangleright\blacktriangleright\curvearrowright}$ that can accept the string $aabcbc$; it surely has to start with reading two times only the symbol a, then it can read two times bc together. However, even in such a case $L(N_{\blacktriangleright\blacktriangleright\curvearrowright}) = \{abcabc, abacbc, aabcbc\}$; which is a proper superset of $\varphi(K)$. Therefore, there cannot be any GJFA $H_{\blacktriangleright\blacktriangleright\curvearrowright}$ that defines $\varphi(K)$. Trivially, φ is also a general homomorphism. The similar proof for $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ is left to the reader. □

Theorem 4.4.17. *All* $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$ *(*$\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$*),* $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$*, and* $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ *are not closed under inverse homomorphism.*

Proof. Consider the language $L = \{aa\}$, which clearly belongs into $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$, $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$. Define the homomorphism $\varphi : \{a\}^* \to \{a\}^*$ as $\varphi(a) = aa$. By applying φ^{-1} to L, we get $\varphi^{-1}(L) = \{a\}$. However, in consequence of Lemma 4.3.2, we know that no 2-jumping mode can define such a language. □

The summary of closure properties of $\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$, $\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$, $\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$, and $\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$ is given in Figure 4.3, where + marks closure, and − marks non-closure.

	$\mathscr{L}_{\blacktriangleright\blacktriangleleft\curvearrowright}$, $\mathscr{L}_{\blacktriangleleft\blacktriangleright\curvearrowright}$	$\mathscr{L}_{\blacktriangleright\blacktriangleright\curvearrowright}$	$\mathscr{L}_{\blacktriangleleft\blacktriangleleft\curvearrowright}$
endmarking (both sides)	− (+)	− (−)	− (−)
concatenation	−	−	−
square (L^2)	−	−	−
shuffle	−	−	−
union	+	+	+
complement	−	−	−
intersection	+	−	−
int. with regular languages	+	−	−
mirror image	+	−	−
finite substitution	−	−	−
homomorphism	+	−	−
ε-free homomorphism	+	−	−
inverse homomorphism	−	−	−

Figure 4.3: Summary of closure properties.

4.5 Concluding Remarks

The resulting behavior of right-left 2-jumps has proven to be very similar to the behaviors of 2-head finite automata accepting linear languages (see [64]) and $5' \to 3'$ sensing Watson-Crick finite automata (see [61, 63, 65]). Although these models differ in details, the general

concept of their reading remains the same—all three mentioned models read simultaneously from two different positions on the opposite sides of the input string. The main difference comes in the form of their rules. The other two models use more complex rules that allow them to read two different strings on their reading positions. Consequently, the resulting language families of these models differ from the language family defined by right-left 2-jumps. Nonetheless, the connection to Watson-Crick models shows that the concept of synchronized jumping could potentially find its use in the fields that study the correlations of several patterns such as biology or computer graphics.

Finally, we propose some future investigation areas concerning jumping finite automata that link several jumps together. Within the previous sections of this chapter, we have already pointed out one open problem concerning right-left (and left-right) 2-jumps (Open Problem 4.3.14). This section continues with other more general suggestions.

(I) Study decidability properties of the newly defined jumping modes.

(II) Investigate remaining possible variants of 2-jumps where the unrestricted single jumps and the restricted single jumps are combined together.

(III) Extend the definition of 2-jumps to the general definition of n-jumps, where $n \geq 1$. Can we find some interesting general results about these multi-jumps?

(IV) Study relaxed versions of 2-jumps where the single jumps do not have to follow the same rule and where each single jump have its own state.

(V) Use the newly defined jumping modes in jumping finite automata in which rules read single symbols rather than whole strings (JFAs—see Section 2.6).

(VI) In the same fashion as in finite automata, consider deterministic versions of GJFAs with the newly defined jumping modes.

Can we have your liver, then?
—Monty Python's The Meaning of Life

Chapter 5

Jumping $5' \to 3'$ Watson-Crick Finite Automata

This chapter studies a combined model of two-head jumping finite automata and sensing $5' \to 3'$ Watson-Crick finite automata. The content of this chapter is composed of results that were published at the conference NCMA 2018 (see [38]) and that are currently submitted to the journal Acta Informatica (see [36]); all written jointly with Zbyněk Křivka, Alexander Meduna, and Benedek Nagy.

In terms of preliminaries, the reader should be familiar with the definitions of general notions (see Section 2.1), grammars (see Sections 2.2.1 and 2.2.2), finite automata (see Section 2.5.1), jumping finite automata (see Section 2.6), and $5' \to 3'$ Watson-Crick finite automata (see Section 2.7).

5.1 Introduction

In recent years, research in formal language theory takes interest in models that process inputs or generate outputs in non-conventional ways compared to classical models of automata and grammars. Due to the direction of the ongoing development in computer science, the main focus is often on models that process information discontinuously and work in a parallel way. In this chapter, we focus our attention on two groups of such models: *jumping finite automata* and *Watson-Crick finite automata*. To be more precise, our main focus is on their specific variants which have multiple heads working in parallel that non-conventionally process the input sequence/string. The studied multi-head variants of finite automaton models utilize two heads that cooperate on a single tape to process the single input string. Therefore, every symbol in the input is read only once, and the heads do not work in the traditional symbol-by-symbol left-to-right way.

To give a better insight into this study, let us briefly introduce both groups of mentioned models. Jumping finite automata were already thoroughly covered in Section 1.2; thus, we only mention a few notes. The jumping concept is in its core focused on discontinuous information processing. Generally, these models can very easily define even some non-context-free languages if the order of symbols can be arbitrary. On the other hand, the resulting language families of these models are usually incomparable with the classical families of regular, linear, and context-free languages. If the jumping concept utilizes multiple heads, the heads can naturally jump on specific positions in the tape, and thus they can easily work on different places at once in parallel.

In DNA computing (see, e.g., [72]), DNA strands can be seen as sequences of nucleotides (there are 4 different nucleotides in the nature, namely Adenine (A), Cytosine (C), Guanine (G), and Thymine (T)). The DNA strands are oriented and they have a $5'$ end and a $3'$ end. Whenever a two-stranded DNA is considered, the two strands are oriented in the opposite way, and the nucleotides at the same position of the two strands must be in Watson-Crick complementarity relation, that is, Adenine appears with Thymine on the other strand and Cytosine appears with Guanine on the other strand. Watson-Crick (WK) finite automata represent a quite settled and already thoroughly studied group of biology-inspired models that can be used to formally process/analyze DNA strands. In essence, a WK automaton works just like a classical finite automaton except it uses a WK tape (i.e., double-stranded tape), and it has a separate head for each of the two strands in the tape. This is therefore a group of models that always naturally use two heads. The classical version of a WK automaton processes the input tape quite conventionally: each head works separately on its own strand of the tape, and both heads read the input in the traditional symbol-by-symbol left-to-right way. However, more recently, new variants of this model were introduced that process the input in a non-conventional way. In a $5' \to 3'$ *WK automaton* (see [60, 61, 63, 65]), both heads read their specific strand in the biochemical $5'$ to $3'$ direction. In a computing point of view, this means that they read the double-stranded sequence in opposite directions. Furthermore, a $5' \to 3'$ WK automaton is *sensing* if the heads sense that they are meeting each other, and the processing of the input ends if for each pair of the sequence one of the letters is read. The sensing $5' \to 3'$ WK automata generally accept the family of linear languages. This concept is also studied further in several follow-up papers that explore alternative definitions and combinations with different mechanics (see [66, 67, 68, 69, 70, 71]).

Even though these two groups significantly differ in their original definitions, their newer models sometimes work in a very similar way. Both concepts are also not mutually exclusive in a single formal model. This chapter defines *jumping* $5' \to 3'$ *WK automata*—a combined model of jumping finite automata and sensing $5' \to 3'$ WK automata—and studies their characteristics. We primarily investigate the accepting power of the model and also the effects of common restrictions on the model.

5.2 Definitions

Considering the sensing $5' \to 3'$ WK automata and full-reading sensing $5' \to 3'$ WK automata described in Section 2.7, there is quite a large gap between their behaviors. On the one hand, sensing $5' \to 3'$ WK automata deliberately read only one of the letters from each complementary pair of the input sequence. However, this also limits the movement of their heads because they can read their strands only until they meet. On the other hand, the definition of full-reading sensing $5' \to 3'$ WK automata allows both heads to traverse their strands completely. Nonetheless, this also means that all complementary pairs of the input sequence are again read twice (as in classical WK automata). If we consider other formal models, we can see that jumping finite automata utilize a mechanism that allows heads to skip (jump over) symbols. Furthermore, some of the introduced double-jumping modes already behave very similar to $5' \to 3'$ WK automata. Due to this natural fit, it is our intention to fill and explore this gap by introducing the jumping mechanism into sensing $5' \to 3'$ WK automata. We want both heads to be able to traverse their strands completely, but we also want to read only one of the letters from each complementary pair of the input sequence.

From a theoretical perspective, the study of such a model is also beneficial for the further understanding of the jumping mechanism. In [57] it was clearly established that the general behavior of jumping finite automaton models is highly nondeterministic. This can be problematic when we try to create viable parsing algorithms based on these models (see [15]). Therefore, there is an interest to study variants of jumping finite automata with more streamlined behavior. Indeed, there are already one-way jumping finite automata with fully deterministic behavior introduced in [7]. Nonetheless, another option is to only limit the amount of possibilities how the heads of the automaton can jump.

With a simple direct approach, it is possible to fit the jumping mechanism straightforwardly into the original definition of sensing $5' \to 3'$ WK automata. (Note that we are now tracking only the meeting event of the heads as it was introduced in [70] and not the original precise sensing distance from [60, 61, 63, 65].)

Definition 5.2.1. A *sensing* $5' \to 3'$ *WK automaton with jumping feature* is a 6-tuple $M = (V, \rho, Q, q_0, F, \delta)$, where V, ρ, Q, q_0, and F are the same as in WK automata, $V \cap \{\#\} = \emptyset$, $\delta\colon (Q \times \binom{V^*}{V^*} \times D) \to 2^Q$, where $D = \{\oplus, \ominus\}$ indicates the mutual position of heads, and $\delta(q, \binom{w_1}{w_2}, s) \neq \emptyset$ only for finitely many quadruples $(q, w_1, w_2, s) \in Q \times V^* \times V^* \times D$. We denote the head as $\blacktriangleright$-head or $\blacktriangleleft$-head if it reads from left to right or from right to left, respectively. We use the symbol $\oplus$ if the $\blacktriangleright$-head is on the input tape positioned before the $\blacktriangleleft$-head; otherwise, we use the symbol $\ominus$. A configuration $\binom{w_1}{w_2}(q, s)\binom{w_1'}{w_2'}$ has the same structure as in sensing $5' \to 3'$ WK automata; however, s indicates only the mutual position of heads, and a partially processed input $\binom{w_1 w_1'}{w_2 w_2'}$ may not satisfy the complementarity ρ. A step of the automaton can be of the following two types: Let $w_1', w_2, x, y, u, v \in V^*$ and $w_1, w_2' \in (V \cup \{\#\})^*$.

(1) Reading steps: $\binom{w_1}{w_2 y}(q, s)\binom{x w_1'}{w_2'} \curvearrowright \binom{w_1 \{\#\}^{|x|}}{w_2}(q', s')\binom{w_1'}{\{\#\}^{|y|} w_2'}$, where $q' \in \delta(q, \binom{x}{y}, s)$, and s' is either $\oplus$ if $|w_2| > |w_1 x|$ or $\ominus$ in other cases.

(2) Jumping steps: $\binom{w_1}{w_2 v}(q, s)\binom{u w_1'}{w_2'} \curvearrowright \binom{w_1 u}{w_2}(q, s')\binom{w_1'}{v w_2'}$, where s' is either $\oplus$ if $|w_2| > |w_1 u|$ or $\ominus$ in other cases.

Note that the jumping steps are an integral and inseparable part of the behavior of the automaton, and thus they are not affected by the state transition function. In the standard manner, let us extend $\curvearrowright$ to $\curvearrowright^n$, where $n \geq 0$; then, based on $\curvearrowright^n$, let us define $\curvearrowright^+$ and $\curvearrowright^*$. The set of all accepted double-stranded strings from $\mathrm{WK}_\rho(V)$, denoted by $LM(M)$, can be defined by the final accepting configurations that can be reached from the initial one: A double-stranded string $\left[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right] \in \mathrm{WK}_\rho(V)$ is accepted by a sensing $5' \to 3'$ WK automaton with jumping feature M if and only if $\binom{\varepsilon}{w_2}(q_0, \oplus)\binom{w_1}{\varepsilon} \curvearrowright^* \binom{w_1'}{\varepsilon}(q_f, \ominus)\binom{\varepsilon}{w_2'}$, for $q_f \in F$, where $w_1' = a_1 a_2 \cdots a_m$, $w_2' = b_1 b_2 \cdots b_m$, $a_i, b_i \in (V \cup \{\#\})$, and either $a_i = \#$ and $b_i \in V$, or $a_i \in V$ and $b_i = \#$, for all $i = 1, \ldots, m$, $m = |w_1|$. The language accepted by M, denoted $L(M)$, is defined as $L(M) = \uparrow_V (LM(M))$.

From a practical point of view, however, this definition is not ideal. The automaton can easily end up in a configuration that cannot yield accepting results, and the correct positions of auxiliary symbols # need to be checked separately at the end of the process. Therefore, we present a modified definition that has the jumping mechanism more integrated into its structure. We are also using a simplification for complementary pairs and treat them as single letters; such a change has no effect on the accepting power, and this form of input is more natural for formal language theory.

Definition 5.2.2. A *jumping* $5' \to 3'$ *WK automaton* is a quintuple $M = (V, Q, q_0, F, \delta)$, where V, Q, q_0, and F are the same as in WK automata, $V \cap \{\#\} = \emptyset$, $\delta \colon (Q \times V^* \times V^* \times D) \to 2^Q$, where $D = \{\oplus, \ominus\}$ indicates the mutual position of heads, and $\delta(q, w_1, w_2, s) \neq \emptyset$ only for finitely many quadruples $(q, w_1, w_2, s) \in Q \times V^* \times V^* \times D$. A configuration (q, s, w_1, w_2, w_3) consists of the state $q \in Q$, the mutual position of heads $s \in D$, and the three unprocessed portions of the input tape: (a) before the first head (w_1), (b) between the heads (w_2), and (c) after the second head (w_3). A step of the automaton can be of the following four types: Let $x, y, u, v, w_2 \in V^*$ and $w_1, w_3 \in (V \cup \{\#\})^*$.

(1) $\oplus$-reading: $(q, \oplus, w_1, xw_2y, w_3) \curvearrowright (q', s, w_1\{\#\}^{|x|}, w_2, \{\#\}^{|y|}w_3)$, where $q' \in \delta(q, x, y, \oplus)$, and s is either $\oplus$ if $|w_2| > 0$ or $\ominus$ in other cases.

(2) $\ominus$-reading: $(q, \ominus, w_1y, \varepsilon, xw_3) \curvearrowright (q', \ominus, w_1, \varepsilon, w_3)$, where $q' \in \delta(q, x, y, \ominus)$.

(3) $\oplus$-jumping: $(q, \oplus, w_1, uw_2v, w_3) \curvearrowright (q, s, w_1u, w_2, vw_3)$, where s is either $\oplus$ if $|w_2| > 0$ or $\ominus$ in other cases.

(4) $\ominus$-jumping: $(q, \ominus, w_1\{\#\}^*, \varepsilon, \{\#\}^*w_3) \curvearrowright (q, \ominus, w_1, \varepsilon, w_3)$.

In the standard manner, let us extend $\curvearrowright$ to $\curvearrowright^n$, where $n \geq 0$; then, based on $\curvearrowright^n$, let us define $\curvearrowright^+$ and $\curvearrowright^*$. The accepted language, denoted by $L(M)$, can be defined by the final accepting configurations that can be reached from the initial one: A string w is accepted by a jumping $5' \to 3'$ WK automaton M if and only if $(q_0, \oplus, \varepsilon, w, \varepsilon) \curvearrowright^* (q_f, \ominus, \varepsilon, \varepsilon, \varepsilon)$, for $q_f \in F$.

Even though the structure of this modified definition is considerably different from Definition 5.2.1, it is not very difficult to show that both models indeed accept the same family of languages.

Lemma 5.2.3. *For every sensing* $5' \to 3'$ *WK automaton with jumping feature* M*, there is a jumping* $5' \to 3'$ *WK automaton* N *such that* $L(M) = L(N)$.

Proof. By construction. Consider any sensing $5' \to 3'$ WK automaton with jumping feature $M = (V, \rho, Q, q_0, F, \delta)$. In Theorem 4 in [42], it was shown that any classical WK automaton $M' = (V', \rho', Q', q_0', F', \delta')$ can be converted into the classical WK automaton $M'' = (V', \rho'', Q', q_0', F', \delta'')$ such that $\rho'' = \{(a, a) : a \in V'\}$ and $L(M') = L(M'')$. The new transition function δ'' is constructed in the following way: For each $q' \in \delta'(q, \binom{u}{v})$ and $\left[\begin{smallmatrix} w \\ v \end{smallmatrix}\right] \in \mathrm{WK}_{\rho'}(V')$, where $q, q' \in Q'$ and $u, v, w \in V'^*$, let $q' \in \delta''(q, \binom{u}{w})$. A similar approach also works for sensing $5' \to 3'$ WK automata. Thus, without loss of generality, assume that δ in M is an identity relation. Let us define the jumping $5' \to 3'$ WK automaton $N = (V, Q, q_0, F, \delta')$, where $\delta'(q, w_1, w_2, s) = \delta(q, \binom{w_1}{w_2}, s)$ for all $q \in Q$, $w_1, w_2 \in V^*$, and $s \in \{\oplus, \ominus\}$.

Now, we show that M and N accept the same language. We say that a current configuration of M is *potentially valid* if M can still potentially reach some accepting configuration $\binom{w_1'}{\varepsilon}(q_f, \ominus)\binom{\varepsilon}{w_2'}$, $q_f \in F$, where $w_1' = a_1a_2 \cdots a_n$, $w_2' = b_1b_2 \cdots b_n$, $a_i, b_i \in (V \cup \{\#\})$, and either $a_i = \#$ and $b_i \in V$, or $a_i \in V$ and $b_i = \#$, for all $i = 1, \ldots, n$, $n = |w_1'|$. Observe that the condition regarding #'s can be checked continuously and individually for each pair $\binom{a_i}{b_i}$ that was already passed by both heads. The following description thus considers only the configurations of M that are still potentially valid.

Let us explore how M can be simulated with N. The accepting process of M can be divided into three distinct stages:

(1) Before the heads meet (the mutual position of heads remains $\oplus$): The reading steps of M clearly correspond with the $\oplus$-reading steps of N—the processed positions are marked with # in both models. Likewise, the jumping steps of M clearly correspond

with the $\oplus$-jumping steps of N—the passed positions are left unchanged for the other head in both models.

(2) The meeting point of heads (when the mutual position changes from $\oplus$ to $\ominus$): The same steps as in (1) are still applicable. The difference is that in M the heads can cross each other, but in N the heads must meet each other precisely. However, the crossing situations in M that lead to potentially valid configurations are quite limited. Assume that the heads of M cross each other, the $\blacktriangleright$-head reads/skips u and the $\blacktriangleleft$-head reads/skips v, then:
   (a) If $|u| > 1$ and $|v| > 1$, the resulting configuration cannot be potentially valid since some pair $\binom{a_i}{b_i}$ was either read or skipped by both heads.
   (b) If $|u| > 1$ and $|v| = 0$: Considering a reading step, all symbols from u that are read after the meeting point must be skipped by the $\blacktriangleleft$-head. However, since jumping steps can occur arbitrarily, there is also an alternative sequence of steps in M where the heads precisely meet, the $\blacktriangleleft$-head jumps afterwards, and the same configuration is reached. Moreover, any jumping step can be replaced with several shorter jumping steps.
   (c) If $|u| = 0$ and $|v| > 1$, the situation is analogous to (b).

   Thus, N does not need to cover these crossing situations.

(3) After the heads met (the mutual position of heads is $\ominus$): To keep the current configuration potentially valid, M can use reading steps only on positions that were not yet read. Correspondingly, N can use $\ominus$-reading steps on positions that do not contain #. Also, M can effectively use jumping steps only on positions that were already read. Correspondingly, N can use $\ominus$-jumping steps on positions that contain #.

From the previous description it is also clear that N cannot accept additional inputs that are not accepted by M since it follows identical state transitions and the steps behave correspondingly between models. Thus, $L(N) = L(M)$. A rigorous version of this proof is rather lengthy but straightforward, so we left it to the reader. □

Lemma 5.2.4. *For every jumping $5' \to 3'$ WK automaton M, there is a sensing $5' \to 3'$ WK automaton with jumping feature N such that $L(M) = L(N)$.*

Proof. By construction. Consider any jumping $5' \to 3'$ WK automaton $M = (V, Q, q_0, F, \delta)$. Let us define the sensing $5' \to 3'$ WK automaton with jumping feature $N = (V, \rho, Q, q_0, F, \delta')$, where $\rho = \{(a, a) : a \in V\}$ and $\delta'(q, \binom{w_1}{w_2}, s) = \delta(q, w_1, w_2, s)$ for all $q \in Q$, $w_1, w_2 \in V^*$, and $s \in \{\oplus, \ominus\}$.

To show that M and N accept the same language, we can follow the reasoning described in the proof of Lemma 5.2.3. The simulation of M with N is trivial since any $\oplus/\ominus$-reading/jumping step of M can be easily simulated with a reading/jumping step of N. Moreover, for the simulated steps, it is guaranteed that the condition regarding #'s holds. Finally, N is clearly not able to accept additional inputs that are not accepted by M. Thus, $L(N) = L(M)$. □

Proposition 5.2.5. *The models of Definitions 5.2.1 and 5.2.2 accept the same family of languages.*

Proof. This proposition follows directly from Lemmas 5.2.3 and 5.2.4. □

Hereafter, we primarily use Definition 5.2.2.

5.3 Examples

To demonstrate the behavior of the automata, we present a few simple examples.

Example 5.3.1. Let us recall that $L = \{w \in \{a,b\}^* : |w|_a = |w|_b\}$ is a well-known nonlinear context-free language. We show that, even though the jumping directions in the model are quite restricted, we are able to accept such a language. Consider the following jumping $5' \to 3'$ WK automaton

$$M = (\{a,b\}, \{s\}, s, \{s\}, \delta)$$

with the state transition function δ: $\delta(s,a,b,\oplus) = \{s\}$ and $\delta(s,a,b,\ominus) = \{s\}$. Starting from s, M can either utilize the jumping or read simultaneously with both heads (the $\blacktriangleright$-head reads a and the $\blacktriangleleft$-head reads b), and it always stays in the sole state s. Now, consider the inputs *aaabbb* and *baabba*. The former can be accepted by using three $\oplus$-readings and one $\ominus$-jumping:

$$(s,\oplus,\varepsilon,aaabbb,\varepsilon) \curvearrowright (s,\oplus,\#,aabb,\#) \curvearrowright (s,\oplus,\#\#,ab,\#\#) \curvearrowright$$
$$(s,\ominus,\#\#\#,\varepsilon,\#\#\#) \curvearrowright (s,\ominus,\varepsilon,\varepsilon,\varepsilon).$$

The latter input is more complex and can be accepted by using one $\oplus$-jumping, two $\oplus$-readings, one $\ominus$-jumping, and one $\ominus$-reading:

$$(s,\oplus,\varepsilon,baabba,\varepsilon) \curvearrowright (s,\oplus,b,aabb,a) \curvearrowright (s,\oplus,b\#,ab,\#a) \curvearrowright$$
$$(s,\ominus,b\#\#,\varepsilon,\#\#a) \curvearrowright (s,\ominus,b,\varepsilon,a) \curvearrowright (s,\ominus,\varepsilon,\varepsilon,\varepsilon).$$

It is not hard to see that, by combining different types of steps, we can accept any input containing the same number of a's and b's, and thus $L(M) = L$.

Example 5.3.2. Consider the following jumping $5' \to 3'$ WK automaton

$$M = (\{a,b\}, \{s\}, s, \{s\}, \delta)$$

with the state transition function δ: $\delta(s,a,b,\oplus) = \{s\}$. Observe that this is almost identical to Example 5.3.1, however, we cannot use the $\ominus$-reading anymore. Consequently, we also cannot effectively use the $\oplus$-jumping because there is no way how to process remaining symbols afterwards. As a result, the accepted language changes to $L(M) = \{a^n b^n : n \geq 0\}$.

Lastly, we give a more complex example that uses all parts of the model.

Example 5.3.3. Consider the following jumping $5' \to 3'$ WK automaton

$$M = (\{a,b,c\}, \{s_0,s_1,s_2\}, s_0, \{s_0\}, \delta)$$

with δ: $\delta(s_0,a,b,\oplus) = \{s_1\}$, $\delta(s_1,\varepsilon,b,\oplus) = \{s_0\}$, $\delta(s_0,c,c,\ominus) = \{s_2\}$, and $\delta(s_2,\varepsilon,c,\ominus) = \{s_0\}$. We can divide the accepting process of M into two stages. First, before the heads meet, the automaton ensures that for every a on the left-hand side there are two b's on the right-hand side; other symbols are skipped with the jumps. Second, after the heads met, the automaton checks if the part before the meeting point has double the number of c's as the part after the meeting point. Thus, $L(M) = \{w_1 w_2 : w_1 \in \{a,c\}^*,\ w_2 \in \{b,c\}^*,\ 2 \cdot |w_1|_a = |w_2|_b,\ |w_1|_c = 2 \cdot |w_2|_c\}$.

5.4 General Results

These results cover the general behavior of jumping $5' \to 3'$ WK automata without any additional restrictions on the model. Let **SWK** and **JWK** denote the language families accepted by sensing $5' \to 3'$ WK automata and jumping $5' \to 3'$ WK automata, respectively.

Considering previous results on other models that use the jumping mechanism (see [57, 58, 31] and Chapters 3 and 4), it is a common characteristic that they define language families that are incomparable with the classical families of regular, linear, and context-free languages. On the other hand, sensing $5' \to 3'$ WK automata (see [60, 61, 63, 65, 70]) are closely related to the family of linear languages. First, we show that the new model is able to accept all linear languages and that its accepting power goes even beyond the family of linear languages.

Lemma 5.4.1. *For every sensing $5' \to 3'$ WK automaton M_1, there is a jumping $5' \to 3'$ WK automaton M_2 such that $L(M_1) = L(M_2)$.*

Proof. This can be proven by construction. Consider any sensing $5' \to 3'$ WK automaton M_1. A direct conversion would be complicated, however, let us recall that **LIN** = **SWK** (see Theorem 2 in [65]). Consider a linear grammar $G = (N, T, P, S)$ such that $L(G) = L(M_1)$. We can construct the jumping $5' \to 3'$ WK automaton M_2 such that $L(M_2) = L(G)$. Assume that $q_f \notin (N \cup T)$. Define $M_2 = (T, N \cup \{q_f\}, S, \{q_f\}, \delta)$, where $B \in \delta(A, u, v, \oplus)$ if $A \to uBv \in P$ and $q_f \in \delta(A, u, \varepsilon, \oplus)$ if $A \to u \in P$ ($A, B \in N$, $u, v \in T^*$).

From the definition of jumping $5' \to 3'$ WK automaton, the $\oplus$-reading steps will always look like this: $(q, \oplus, w_1, uw_2v, w_3) \curvearrowright (q', s, w_1\{\#\}^{|u|}, w_2, \{\#\}^{|v|}w_3)$, where $q' \in \delta(q, u, v, \oplus)$, $w_2 \in T^*$, $w_1, w_3 \in (T \cup \{\#\})^*$, and s is either $\oplus$ if $|w_2| > 0$ or $\ominus$ in other cases. In M_2, there are no possible $\ominus$-reading steps. The $\oplus$-jumping can be potentially used to skip some symbols before the heads meet; nonetheless, this leads to the configuration (q, s, w_1, w_2, w_3) where $\text{alph}(w_1w_3) \cap T \neq \emptyset$. Since without $\ominus$-reading steps there is no way how to read symbols of T in w_1 and w_3, such a configuration cannot yield an accepting result. Consequently, starting from $(S, \oplus, \varepsilon, w, \varepsilon)$ where $w \in T^*$, it can be easily seen that if M_2 accepts w, it reads all symbols of w in the same fashion as G generates them; the remaining #'s can be erased with the $\ominus$-jumping afterwards. Moreover, the heads of M_2 can meet each other with the accepting state q_f if and only if G can finish the generation process with a rule $A \to u$. Thus, $L(M_2) = L(G) = L(M_1)$. □

Theorem 5.4.2. ***LIN* = *SWK* ⊂ *JWK*.**

Proof. **SWK** ⊆ **JWK** follows from Lemma 5.4.1. **LIN** = **SWK** was proven in [65]. **JWK** ⊈ **LIN** follows from Example 5.3.1. □

The next two characteristics follow from the previous results.

Theorem 5.4.3. *Jumping $5' \to 3'$ WK automata that do not use $\ominus$-reading steps accept linear languages.*

Proof. Consider any jumping $5' \to 3'$ WK automaton $M = (V, Q, q_0, F, \delta)$ that has no possible $\ominus$-reading steps. Following the reasoning from the proof of Lemma 5.4.1, if there are no possible $\ominus$-reading steps, the $\oplus$-jumping cannot be effectively used, and we can construct a linear grammar that generates strings in the same fashion as M reads them. Define the linear grammar $G = (Q, V, R, q_0)$, where R is constructed in the following way: (1) For each $p \in \delta(q, u, v, \oplus)$, add $q \to upv$ to R. (2) For each $f \in F$, add $f \to \varepsilon$ to R. Clearly, $L(G) = L(M)$. □

Proposition 5.4.4. *The language family accepted by double-jumping finite automata that perform right-left and left-right jumps (see Chapter 4) is strictly included in* ***JWK****.*

Proof. First, Theorem 4.3.18 shows that jumping finite automata that perform right-left and left-right jumps accept the same family of languages. Second, Theorem 4.3.7 shows that this family is strictly included in **LIN**. Finally, Theorem 5.4.2 shows that **LIN** is strictly included in **JWK**. □

Even though jumping $5' \to 3'$ WK automata are able to accept some nonlinear languages, the jumping directions of their heads are quite restricted compared to general jumping finite automata. Consequently, there are some languages accepted by jumping $5' \to 3'$ WK automata and general jumping finite automata that cannot be accepted with the other model. To formally prove these results, we need to introduce the concept of the debt of a configuration in jumping $5' \to 3'$ WK automata. First, we start with the formal definition of a reachable state.

Definition 5.4.5. Let $M = (V, Q, q_0, F, \delta)$ be a jumping $5' \to 3'$ WK automaton. Assuming some states $q, q' \in Q$ and a mutual position of heads $s \in \{\oplus, \ominus\}$, we say that q' is *reachable* from q and s if there exists a configuration (q, s, w_1, w_2, w_3) such that $(q, s, w_1, w_2, w_3) \curvearrowright^* (q', s', w_1', w_2', w_3')$ in M, $s' \in \{\oplus, \ominus\}$, $w_1, w_2, w_3, w_1', w_2', w_3' \in (V \cup \{\#\})^*$.

Next, we show that for any computation $\mathcal{C}$ that takes a jumping $5' \to 3'$ WK automaton M from a starting configuration to a configuration from which a final state is reachable, there exists $w' \in L(M)$ such that w' can be fully processed with the same sequence of reading steps as in $\mathcal{C}$ and a limited number of additional steps. Note that jumping steps in $\mathcal{C}$ are unimportant for the result since jumping steps can occur arbitrarily and they do not process any symbols of the input.

Lemma 5.4.6. *For each jumping $5' \to 3'$ WK automaton $M = (V, Q, q_0, F, \delta)$ there exists a constant k such that the following holds. Let $q \in Q$ and $s \in \{\oplus, \ominus\}$ such that $f \in F$ is reachable from q and s. For every computation $\mathcal{C}$ that takes M from $(q_0, \oplus, \varepsilon, w, \varepsilon)$ to (q, s, w_1, w_2, w_3), $w \in V^*$, $w_1, w_2, w_3 \in (V \cup \{\#\})^*$, there exists $w' \in L(M)$ such that M starting with w' can reach q and $s' \in \{\oplus, \ominus\}$ by using the same sequence of $\oplus/\ominus$-reading steps as in $\mathcal{C}$ and the rest of w' can be processed with a limited number of additional steps bounded by k.*

Proof. First, if f is reachable from q and s, then there exists a sequence of pairs $\mathcal{P} = (p_0, s_0) \cdots (p_n, s_n)$, for some $n \geq 0$, where

- $p_i \in Q$, $s_i \in \{\oplus, \ominus\}$, for all $i = 0, \ldots, n$,
- $p_0 = q$, $s_0 = s$ or $s_0 = \ominus$, $p_n = f$, $s_n = \ominus$,
- for all $i = 0, \ldots, n-1$ it holds: $p_{i+1} \in \delta(p_i, x_i, y_i, s_i)$, $x_i, y_i \in V^*$,
- for all $i = 0, \ldots, n-1$ it holds: $s_{i+1} = s_i$ or $s_{i+1} = \ominus$, and
- $(p_i, s_i) = (p_j, s_j)$ implies $i = j$, $i, j = 0, \ldots, n$ (all pairs are unique).

This sequence is finite, and its maximum length is bounded by $k' = 2 \cdot |Q|$.

Second, let us represent a $\oplus/\ominus$-reading step as a quintuple (q', x, y, s'', q'') according to $q'' \in \delta(q', x, y, s'')$, $q', q'' \in Q$, $x, y \in V^*$, $s'' \in \{\oplus, \ominus\}$. From the computation $\mathcal{C}$ we extract a sequence of $\oplus/\ominus$-reading steps $\mathcal{S}$. From the sequence of pairs $\mathcal{P}$ we can easily derive a sequence of $\oplus/\ominus$-reading steps $\mathcal{S}'$ that follows the state transitions of $\mathcal{P}$. Let $\mathcal{S}'' = \mathcal{S}\mathcal{S}'$. We split $\mathcal{S}''$ into two parts $\mathcal{S}'' = \mathcal{S}''_{\oplus}\mathcal{S}''_{\ominus}$ such that $\mathcal{S}''_{\oplus} = (p_0', a_0, b_0, \oplus, q_0') \cdots (p_n', a_n, b_n, \oplus, q_n')$ and $\mathcal{S}''_{\ominus} = (p_0'', c_0, d_0, \ominus, q_0'') \cdots (p_m'', c_m, d_m, \ominus, q_m'')$, where $n, m \geq 0$, $i = 0, \ldots, n$, $j = 0, \ldots, m$, $p_i', q_i', p_j'', q_j'' \in Q$, $a_i, b_i, c_j, d_j \in V^*$.

Third, we consider input $w' = a_0 \cdots a_n d_m \cdots d_0 c_0 \cdots c_m b_n \cdots b_0$. It is not hard to construct a computation $\mathcal{C}'$ of M from $\mathcal{S}''_{\oplus}$, one $\oplus$-jumping step, $\mathcal{S}''_{\ominus}$, and one $\ominus$-jumping step such that

$$
\begin{aligned}
&(q_0, \oplus, \varepsilon, a_0 \cdots a_n d_m \cdots d_0 c_0 \cdots c_m b_n \cdots b_0, \varepsilon) \curvearrowright^* \\
&(q', s''', \{\#\}^{|a_0 \cdots a_n|}, d_m \cdots d_0 c_0 \cdots c_m, \{\#\}^{|b_n \cdots b_0|}) \curvearrowright \\
&(q', \ominus, \{\#\}^{|a_0 \cdots a_n|} d_m \cdots d_0, \varepsilon, c_0 \cdots c_m \{\#\}^{|b_n \cdots b_0|}) \curvearrowright^* \\
&(f, \ominus, \{\#\}^{|a_0 \cdots a_n|}, \varepsilon, \{\#\}^{|b_n \cdots b_0|}) \curvearrowright \\
&(f, \ominus, \varepsilon, \varepsilon, \varepsilon),
\end{aligned}
$$

$q' \in Q$, $s''' \in \{\oplus, \ominus\}$. Thus, $w' \in L(M)$ and there exists $k \leq k'$ for M that bounds the number of additional steps. □

Next, based on known M and $L(M)$, we can define the debt of a configuration of M. If we follow a computation of M on an input w, we can easily determine the Parikh vector o of symbols already processed from w in a current configuration γ. Additionally, with the known $L(M)$, we can determine Parikh vectors for all $w' \in L(M)$. The debt of the configuration γ represents the minimum number of symbols that have to be added to o so that o matches the Parikh vector of some $w' \in L(M)$. Note that we use ∞ to cover situations when no match is possible.

Definition 5.4.7. Let $M = (V, Q, q_0, F, \delta)$ be a jumping $5' \to 3'$ WK automaton, where $V = \{a_1, \ldots, a_n\}$, and let $w \in V^*$. We define the Parikh vector $o = (o_1, \ldots, o_n)$ of processed (read) symbols from w in a configuration $\gamma = (q, s, w_1, w_2, w_3)$ of M reached from an initial configuration $(q_0, \oplus, \varepsilon, w, \varepsilon)$ of M as $o = \Psi_V(w) - \Psi_V(w_1 w_2 w_3)$, $q \in Q$, $s \in \{\oplus, \ominus\}$, $w_1, w_2, w_3 \in (V \cup \{\#\})^*$. Using the Parikh mapping of $L(M)$, we define $\Delta(o) = \{\sum_{i=1}^{n}(m_i - o_i) : (m_1, \ldots, m_n) \in \Psi_V(L(M)),\ m_i \geq o_i,\ 1 \leq i \leq n\} \cup \{\infty\}$. Finally, we define the *debt* of the configuration γ of M as $\min \Delta(o)$.

And finally, we can combine Lemma 5.4.6 and Definition 5.4.7 to show that each jumping $5' \to 3'$ WK automaton M has to accept all $w \in L(M)$ over configurations with some bounded debt.

Lemma 5.4.8. *Let L be a language, and let $M = (V, Q, q_0, F, \delta)$ be a jumping $5' \to 3'$ WK automaton. If $L(M) = L$, there exists a constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k.*

Proof. By contradiction. Assume that there is no constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k. Then, M can accept some $w \in L$ over a configuration for which the debt cannot be bounded by any k. Let $V = \{a_1, \ldots, a_n\}$. Consider any configuration γ of M reached from an initial configuration $(q_0, \oplus, \varepsilon, w, \varepsilon)$ of M. Let $o = (o_1, \ldots, o_n)$ be the Parikh vector of processed symbols from w in γ. First, assume that γ contains a state $q \in Q$ with a mutual position of heads $s \in \{\oplus, \ominus\}$ from which a final state $f \in F$ is reachable. Then, due to Lemma 5.4.6, there is $w' \in L(M)$ such that $\Psi(w') = (m_1, \ldots, m_n)$, $m_i \geq o_i$, $1 \leq i \leq n$, and $|w'| \leq \sum_{i=1}^{n}(o_i) + k'$, where k' is some constant for M. According to Definition 5.4.7, $w' \in L(M)$ implies $\min \Delta(o) \leq k'$. Second, assume that γ contains a state q with a mutual position of heads s from which no final state f is reachable. Then, by Definitions 5.2.2 and 5.4.5, there is no computation that takes M from γ to a final accepting configuration. Thus, when M accepts w, it must

be done over configurations with the debt $\leq k'$. However, that is a contradiction with the assumption that M can accept some $w \in L$ over a configuration for which the debt cannot be bounded by any k. □

Observe that the debt alone does not depend on the order of symbols in the words of $L(M)$, e.g., $\Psi_V(\{(abc)^n : n \geq 0\}) = \Psi_V(\{a^n b^n c^n : n \geq 0\})$, for $V = \{a, b, c\}$. However, when the debt is combined with the computational possibilities of M on an input w, we can show that a language L cannot be accepted by M if there is no k for M such that all $w \in L$ can be fully processed over configurations of M with the debt bounded by k.

Lemma 5.4.9. *There is no jumping* $5' \to 3'$ *WK automaton* M *such that* $L(M) = \{a^n b^n c^n : n \geq 0\}$.

Proof. Basic idea. Considering any sufficiently large constant k, we show that M cannot process all symbols of $a^{10k}b^{10k}c^{10k}$ using only configurations that have their debt bounded by k.
Formal proof. By contradiction. Let $L = \{a^n b^n c^n : n \geq 0\}$, and let $M = (V, Q, q_0, F, \delta)$ be a jumping $5' \to 3'$ WK automaton such that $L(M) = L$. Due to Lemma 5.4.8, there must exist a constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k. (Observe that if Lemma 5.4.8 holds for some constant k', it also holds for all $k'' > k'$.) Let $k_{\min} = \max\{|uv| : \delta(q, u, v, s) \neq \emptyset,\ q \in Q,\ u, v \in V^*,\ s \in \{\oplus, \ominus\}\}$. Consider any k for M such that $k > k_{\min}$. Due to the structure of L, we can represent the debt of the configuration of M as $\langle d_a, d_b, d_c\rangle$, where d_a, d_b, d_c is the minimum number of symbols a, b, c that M must yet to read to get the balanced number of processed symbols. (For illustration, an initial configuration of M has the debt $\langle 0, 0, 0\rangle$. When M reads a, the following configuration has the debt $\langle 0, 1, 1\rangle$ because at least one b and one c have yet to be read to keep the number of processed symbols balanced.) When $(q_0, \oplus, \varepsilon, w, \varepsilon) \curvearrowright^* (q_f, \ominus, \varepsilon, \varepsilon, \varepsilon)$ in M, $q_f \in F$, for all traversed configurations must hold $d_a + d_b + d_c \leq k$. Let $w = a^{10k}b^{10k}c^{10k}$.

First, we explore the maximum number of symbols that M can read from w before the heads meet. Starting from the initial configuration $(q_0, \oplus, \varepsilon, w, \varepsilon)$ with the debt $\langle 0, 0, 0\rangle$ and until the mutual position $\ominus$ is reached, M can use $\oplus$-reading steps to process symbols and $\oplus$-jumping steps to skip symbols. Consider different reading strategies that try to process the maximum number of symbols from $a^{10k}b^{10k}c^{10k}$ before the heads meet. There are three distinct places where the heads of M can meet:

(A) Assume that the heads meet inside the segment of a's:
  (1) M can process (with multiple steps) a^k and c^k until it reaches the debt $\langle 0, k, 0\rangle$. Then, M has to start read b's.
  (2) M can read l symbols together in one step (balanced number of a's, b's, and c's) while keeping the debt $\langle 0, k, 0\rangle$, $l < k$. Nonetheless, the $\blacktriangleleft$-head ends up in the segment of b's.
  (3) M can process a^k and b^{2k} until it reaches the debt $\langle 0, 0, k\rangle$. Clearly, there is no way how to read additional c's.

  No further reading is possible, and this strategy can process $5k + l$ symbols.

(B) Assume that the heads meet inside the segment of c's. Then, this is just a mirror case of (A), and this strategy can process $5k + l$ symbols.

(C) Assume that the heads meet inside the segment of b's. Observe that in (A) and (B) the heads can meet on the border of a's and b's or b's and c's. There are no additional possibilities when both heads read b's since the debt is limited by the letters that were

already skipped by one of the heads. Thus, this strategy can process $5k + l$ symbols as well.

Consequently, before the heads meet, M can process no more than $5k + l$ symbols.

Second, when the heads meet, $a^{>4k}b^{>4k}c^{>4k}$ has yet to be processed. The heads are next to each other, and M can use $\ominus$-reading steps to process symbols and $\ominus$-jumping steps to remove the auxiliary #'s. Consider different reading strategies that try to process the maximum number of symbols after the heads met. There are several distinct places where the heads of M can be positioned:

(A) Assume that the heads are between a's and b's. It is possible to start with a debt up to k. Consider the debt $\langle 0, k, 0\rangle$. M can process a^k and b^{2k} until it reaches the debt $\langle 0, 0, k\rangle$. Since there is $b^{>4k}$, it is not possible to reach c's. Clearly, it is not possible to select a different debt that would yield a better result. Thus, this strategy can process $3k$ symbols.

(B) Assume that the heads are between b's and c's. Then, this is just a mirror case of (A), and this strategy can process $3k$ symbols.

(C) Assume that the heads are in the middle of b's with the debt $\langle 0, k, 0\rangle$. M can process $b^{\frac{3}{2}k}$ until it reaches $\langle \frac{1}{2}k, 0, \frac{1}{2}k\rangle$. It is not possible to reach neither a's or c's. Thus, this strategy can process $\frac{3}{2}k$ symbols.

(D) Any other position of the heads can be seen as a slightly modified case of (A), (B), or (C). Since neither of these cases is able to reach all three types of symbols, they can process only up to $3k$ symbols.

Consequently, after the heads met, M can process no more than $3k$ symbols.

Finally, we can see that M is not able to process more than $8k + l$ symbols from $w = a^{10k}b^{10k}c^{10k}$ when the debt of configurations of M is bounded by k. Since, for any k, $w \in L$ and w contains $30k$ symbols, there is no constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k. But that is a contradiction with the assumption that there is a jumping $5' \to 3'$ WK automaton M such that $L(M) = \{a^n b^n c^n : n \geq 0\}$. □

Lemma 5.4.10. *There is no jumping $5' \to 3'$ WK automaton M such that $L(M) = \{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$.*

Proof. Let $N = (V, Q, q_0, F, \delta)$ be a jumping $5' \to 3'$ WK automaton, $L = \{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$, and $K = \{a^n b^n c^n : n \geq 0\}$. Let w be an input of the form $a^*b^*c^*$. Let γ be a configuration of N reached from an initial configuration $(q_0, \oplus, \varepsilon, w, \varepsilon)$ of N. Let o be the Parikh vector of processed symbols from w in γ. Observe that, for any γ, the debt of the configuration $\min \Delta(o)$ is similar for $L(N) = L$ and $L(N) = K$ since it only depends on o and the quantities of symbols in the words of the language $L(N)$. Consequently, the proof that there is no such M is analogous to the proof of Lemma 5.4.9. □

Proposition 5.4.11. ***JWK*** *is incomparable with* ***GJFA*** *and* ***JFA****.*

Proof. The language $\{w \in \{a, b\}^* : |w|_a = |w|_b\}$ from Example 5.3.1 and the language $\{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$ from Lemma 5.4.10 are accepted with (general) jumping finite automata (see Example 5 in [57]). The language $\{a^n b^n : n \geq 0\}$ from Example 5.3.2 is not accepted with (general) jumping finite automata (see Lemma 19 in [57]). □

The last group of results compares the accepting power of jumping $5' \to 3'$ WK automata with the families of context-sensitive and context-free languages.

Theorem 5.4.12. ***JWK*** $\subset$ ***CS***.

Proof. Clearly, the use of two heads and the jumping behavior can be simulated by linear bounded automata, so **JWK** $\subseteq$ **CS**. From Lemma 5.4.9, **CS** $-$ **JWK** $\neq \emptyset$. □

Lemma 5.4.13. *There are some non-context-free languages accepted by jumping* $5' \to 3'$ *WK automata.*

Proof. Consider the following jumping $5' \to 3'$ WK automaton

$$M = (\{a, b, c, d\}, \{s\}, s, \{s\}, \delta)$$

with the state transition function δ: $\delta(s, a, c, \oplus) = \{s\}$ and $\delta(s, d, b, \ominus) = \{s\}$. The accepting process has two stages. First, before the heads meet, the automaton reads the same number of a's and c's. Second, after the heads met, the automaton reads the same number of d's and b's. Thus, $L(M) = \{w_1 w_2 : w_1 \in \{a, b\}^*,\ w_2 \in \{c, d\}^*,\ |w_1|_a = |w_2|_c,\ |w_1|_b = |w_2|_d\}$. By contradiction. Assume that $L(M)$ is a context-free language. The family of context-free languages is closed under intersection with regular sets. Let $K = L(M) \cap \{a\}^*\{b\}^*\{c\}^*\{d\}^*$. Clearly, there are some strings in $L(M)$ that satisfy this forced order of symbols. Furthermore, they all have the proper correlated numbers of these symbols. Consequently, $K = \{a^n b^m c^n d^m : n, m \geq 0\}$. However, K is a well-known non-context-free language (see Chapter 3.1 in [78]). That is a contradiction with the assumption that $L(M)$ is a context-free language. Therefore, $L(M)$ is a non-context-free language. □

Lemma 5.4.14. *There is no jumping* $5' \to 3'$ *WK automaton* M *such that* $L(M) = \{a^n b^n c^m d^m : n, m \geq 0\}$.

Proof. Basic idea. We follow the proof structure of Lemma 5.4.9. Considering any sufficiently large constant k, we show that M cannot process all symbols of $a^{10k} b^{10k}\ c^{10k} d^{10k}$ using only configurations that have their debt bounded by k.
Formal proof. By contradiction. Let $L = \{a^n b^n c^m d^m : n, m \geq 0\}$, and let $M = (V, Q, q_0, F, \delta)$ be a jumping $5' \to 3'$ WK automaton such that $L(M) = L$. Due to Lemma 5.4.8, there must exist a constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k. Let $k_{\min} = \max\{|uv| : \delta(q, u, v, s) \neq \emptyset,\ q \in Q,\ u, v \in V^*,\ s \in \{\oplus, \ominus\}\}$. Consider any k for M such that $k > k_{\min}$. Let $w = a^{10k} b^{10k} c^{10k} d^{10k}$.

We follow a computation of M from an initial configuration $\sigma = (q_0, \oplus, \varepsilon, w, \varepsilon)$. First, we explore the limits of how many symbols M can read with $\ominus$-reading steps. Let γ be the first configuration of the computation of M with the mutual position of heads $\ominus$ reached from σ. Consider the maximum number of b's that the $\blacktriangleleft$-head can read with $\ominus$-reading steps starting from γ. Since a's are in front of b's and since a's are linked with b's, this number must be limited. The configuration γ can have the debt of at most k b's, the debt can reach at most k a's, and only one step can read both types of symbols together. Thus, the maximum number of b's that the $\blacktriangleleft$-head can read with $\ominus$-reading steps starting from γ is less than $3k$. In the same manner, the maximum number of c's that the $\blacktriangleright$-head can read with $\ominus$-reading steps starting from γ is less than $3k$.

Second, we explore the limits of how many symbols M can read with $\oplus$-reading steps. Consider the maximum number of a's and b's that the $\blacktriangleright$-head can read on its own with $\oplus$-reading steps starting from γ. The configuration σ has no debt, the debt can reach at most k b's, only one step can read both types of symbols together, and then the debt can reach at most k a's. Thus, the maximum number of a's and b's that the $\blacktriangleleft$-head can read

on its own with $\oplus$-reading steps starting from γ is less than $4k$. In the same manner, the maximum number of c's and d's that the $\blacktriangleleft$-head can read on its own with $\oplus$-reading steps starting from γ is less than $4k$. Due to the previous limits with $\ominus$-reading steps, in a successful computation, the $\blacktriangleright$-head cannot jump over all remaining b's and the $\blacktriangleleft$-head cannot jump over all remaining c's. Thus, the heads cannot meet in a configuration that can yield a successful computation.

Finally, we can see that M is not able to accept $w = a^{10k}b^{10k}c^{10k}d^{10k}$ when the debt of configurations of M is bounded by k. Since, for any k, $w \in L$, there is no constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k. But that is a contradiction with the assumption that there is a jumping $5' \to 3'$ WK automaton M such that $L(M) = \{a^n b^n c^m d^m : n, m \geq 0\}$. □

Theorem 5.4.15. ***JWK*** *and* ***CF*** *are incomparable.*

Proof. **JWK** $\not\subseteq$ **CF** follows from Lemma 5.4.13. **CF** $\not\subseteq$ **JWK** follows from Lemma 5.4.14. Lastly, **LIN** $\subset$ **JWK** and **LIN** $\subset$ **CF**. □

5.5 Results on Restricted Variants

In this section, we compare the accepting power of unrestricted and restricted variants of jumping $5' \to 3'$ WK automata. This paper considers the same standard restrictions as they are defined for Watson-Crick finite automata. Since these restrictions regulate only the state control and reading steps of the automaton, the jumping is not affected in any way. Let **JWK** denote the language family accepted by jumping $5' \to 3'$ WK automata. We are using prefixes **N**, **F**, **S**, **1**, **NS**, **FS**, **N1**, and **F1** to specify the restricted variants of jumping $5' \to 3'$ WK automata and appropriate language families.

In [57] it was shown that the use of the jumping mechanism can have an unusual impact on the expressive power of the automaton model when we restrict the state transition function of the model. In the case of classical finite automata, it makes no difference if the steps of the automaton read single symbols or longer strings. Nonetheless, in the case of jumping finite automata, this change has a large impact on the expressive power of the model. Moreover, most of the standard restrictions studied in Watson-Crick finite automata were not yet considered together with the jumping mechanism. Therefore, it is our intention to thoroughly explore the impact of these restrictions on the accepting power of jumping $5' \to 3'$ WK automata and compare it with the similar results obtained for sensing $5' \to 3'$ WK automata.

In the field of DNA computing, the empty string/sequence usually does not belong to any language because it does not refer to a molecule. This paper is not so strict and thus considers the empty string as a possible valid input. Nonetheless, the following proofs are deliberately based on more complex inputs to mitigate the impact of the empty string on the results. Moreover, we distinguish between **FIN** and $\mathbf{FIN}_{\varepsilon\text{-inc}}$, when the difference is unavoidable.

Note that there are some inherent inclusions between language families based on the application of restrictions on the model. Additionally, several other basic relations can be established directly from the definitions of the restrictions:

Lemma 5.5.1. *The following relations hold: (i)* ***N JWK*** $\subseteq$ ***F JWK****; (ii)* ***1 JWK*** $\subseteq$ ***S JWK****; (iii)* ***F1 JWK*** $\subseteq$ ***FS JWK****; (iv)* ***N1 JWK*** $\subseteq$ ***NS JWK****; (v)* ***NS JWK*** $\subseteq$ ***FS JWK****; (vi)* ***N1 JWK*** $\subseteq$ ***F1 JWK****.*

Proof. These results follow directly from the definitions since the stateless restriction (**N**) is a special case of the all-final restriction (**F**) and the 1-limited restriction (**1**) is a special case of the simple restriction (**S**). □

5.5.1 Simple Restriction

Theorem 5.5.2. ***S JWK* = *JWK*.**

Proof. Basic idea. Any general reading step can be replaced with at most two simple reading steps and a new auxiliary state that together accomplish the same action.
Formal proof. By construction. Consider any jumping $5' \to 3'$ WK automaton $M = (V, Q_1, q_0, F, \delta_1)$. We can construct the **S** jumping $5' \to 3'$ WK automaton N such that $L(N) = L(M)$. Define $N = (V, Q_2, q_0, F, \delta_2)$, where Q_2 and δ_2 are created in the following way: Let $q \in Q_1$, $x, y \in V^*$, and $s \in \{\oplus, \ominus\}$.
(1) Set $Q_2 = Q_1$.
(2) For each $\delta_1(q, x, y, s) \neq \emptyset$ where $|x| = 0$ or $|y| = 0$,
let $\delta_2(q, x, y, s) = \delta_1(q, x, y, s)$.
(3) For each $\delta_1(q, x, y, s) \neq \emptyset$ where $|x| > 0$ and $|y| > 0$, add a new unique state p to Q_2 and let $p \in \delta_2(q, x, \varepsilon, s)$ and $\delta_2(p, \varepsilon, y, s) = \delta_1(q, x, y, s)$.
It is clear that all original transitions that did not satisfy the simple restriction were transformed into the new suitable transitions.

Now we show that this change has no effect on the accepted language. Let $w \in L(M)$ be accepted by an accepting computation γ of M. There is a computation γ' of N corresponding to γ of M. We can construct γ' from γ in the following way:
(A) If there is $(q, \oplus, w_1, xw_2y, w_3) \curvearrowright (q', s, w_1\{\#\}^{|x|}, w_2, \{\#\}^{|y|}, w_3)$ in γ, where $x, y \in V^+$, $w_1, w_2, w_3 \in (V \cup \{\#\})^*$, $q, q' \in Q_1$, $s \in \{\oplus, \ominus\}$, we replace it in γ' with $(q, \oplus, w_1, xw_2y, w_3) \curvearrowright (p, \oplus, w_1\{\#\}^{|x|}, w_2y, w_3) \curvearrowright (q', s, w_1\{\#\}^{|x|}, w_2, \{\#\}^{|y|}w_3)$, where p is the new state introduced for $\delta_1(q, x, y, \oplus)$ in step (3).
(B) If there is $(q, \ominus, w_1y, \varepsilon, xw_3) \curvearrowright (q', \ominus, w_1, \varepsilon, w_3)$ in γ, where $x, y \in V^+$, $w_1, w_3 \in (V \cup \{\#\})^*$, $q, q' \in Q_1$, we replace it in γ' with $(q, \ominus, w_1y, \varepsilon, xw_3) \curvearrowright (p, \ominus, w_1y, \varepsilon, w_3) \curvearrowright (q', \ominus, w_1, \varepsilon, w_3)$, where p is the new state introduced for $\delta_1(q, x, y, \ominus)$ in step (3).
(C) We keep other steps of the computation without changes.
Clearly, γ' is an accepting computation of N and $w \in L(N)$. Thus, $L(M) \subseteq L(N)$.

Let $w \in L(N)$ be accepted by an accepting computation γ of N. Clearly, any sequence of consecutive $\oplus/\ominus$-jumping steps can be replaced with a single $\oplus/\ominus$-jumping step, and it is also possible to utilize empty jumping steps that do not move the heads. Thus, without loss of generality, assume that γ does not contain sequences of consecutive $\oplus/\ominus$-jumping steps and that every reading step in γ is followed by a jumping step. There is a computation γ' of M corresponding to γ of N. We can construct γ' from γ in the following way:
(A) If there is $(q, \oplus, w_1, xuw_2yv, w_3) \curvearrowright (p, \oplus, w_1\{\#\}^{|x|}, uw_2yv, w_3) \curvearrowright (p, \oplus, w_1\{\#\}^{|x|}u, w_2y, vw_3) \curvearrowright (q', s, w_1\{\#\}^{|x|}u, w_2, \{\#\}^{|y|}vw_3)$ in γ, where $x, y \in V^+$, $u, v \in V^*$, $w_1, w_2, w_3 \in (V \cup \{\#\})^*$, $q, q' \in Q_1$, $s \in \{\oplus, \ominus\}$, and p is the new state introduced for $\delta_1(q, x, y, \oplus)$ in step (3), we replace these steps in γ' with $(q, \oplus, w_1, xuw_2yv, w_3) \curvearrowright (q, \oplus, w_1, xuw_2y, vw_3) \curvearrowright (q', s', w_1\{\#\}^{|x|}, uw_2, \{\#\}^{|y|}vw_3) \curvearrowright (q', s, w_1\{\#\}^{|x|}u, w_2, \{\#\}^{|y|}vw_3)$, where $s' \in \{\oplus, \ominus\}$ according to the definition of $\oplus$-reading steps. Observe that, due to the unique p, it is clear that $q' \in \delta_1(q, x, y, \oplus)$ in M.
(B) If there is $(q, \ominus, w_1y\{\#\}^v, \varepsilon, x\{\#\}^uw_3) \curvearrowright (p, \ominus, w_1y\{\#\}^v, \varepsilon, \{\#\}^uw_3) \curvearrowright (p, \ominus, w_1y, \varepsilon, w_3) \curvearrowright (q', \ominus, w_1, \varepsilon, w_3)$ in γ, where $x, y \in V^+$, $u, v \geq 0$, $w_1, w_3 \in (V \cup \{\#\})^*$, $q, q' \in Q_1$, and p is the new state introduced for $\delta_1(q, x, y, \ominus)$ in step (3), we re-

place these steps in γ' with $(q, \ominus, w_1 y\{\#\}^v, \varepsilon, x\{\#\}^u w_3) \curvearrowright (q, \ominus, w_1 y, \varepsilon, x\{\#\}^u w_3) \curvearrowright (q', \ominus, w_1, \varepsilon, \{\#\}^u w_3) \curvearrowright (q', \ominus, w_1, \varepsilon, w_3)$.

(C) We keep other steps of the computation without changes.

Clearly, γ' is an accepting computation of M and $w \in L(M)$. Thus, $L(N) \subseteq L(M)$. Consequently, $L(N) = L(M)$. □

5.5.2 1-limited Restriction

Example 5.5.3. Consider the following jumping $5' \to 3'$ WK automaton $M = (\{a, b, c\}, \{s, f\}, s, \{f\}, \delta)$ with the state transition function δ:

$$\delta(s, a, b, \oplus) = \{s\}, \quad \delta(f, a, b, \oplus) = \{f\}, \quad \delta(f, a, b, \ominus) = \{f\},$$
$$\delta(s, cc, \varepsilon, \oplus) = \{f\}, \quad \delta(s, \varepsilon, cc, \oplus) = \{f\}.$$

The first three transitions mimic the behavior of Example 5.3.1. The other two transitions ensure that the input is accepted only if it also contains precisely one substring cc. Therefore, $L(M) = \{w_1 ccw_2 : w_1, w_2 \in \{a, b\}^*, \ |w_1 w_2|_a = |w_1 w_2|_b\}$.

Lemma 5.5.4. *Let $M = (V, Q, q_0, F, \delta)$ be a **1** jumping $5' \to 3'$ WK automaton M, and let $w \in L(M)$ be accepted by an accepting computation γ of M. Let us represent the $\oplus$-reading step of M that follows $\delta(q, u, v, \oplus)$, $q \in Q$, $u, v \in (V \cup \{\varepsilon\})$, as a quadruple $(u, v, \varepsilon, \varepsilon)$ and the $\ominus$-reading step of M that follows $\delta(q, u', v', \ominus)$, $q \in Q$, $u', v' \in (V \cup \{\varepsilon\})$, as a quadruple $(\varepsilon, \varepsilon, u', v')$. Then, we can represent the reading steps of γ as a sequence $(u_1, v_1, u'_1, v'_1) \cdots (u_n, v_n, u'_n, v'_n)$, $i = 1, \ldots, n$, $n \geq 1$. Let $w_{\blacktriangleright\oplus} = u_1 \cdots u_n$, $w_{\blacktriangleright\ominus} = u'_1 \cdots u'_n$, $w_{\blacktriangleleft\oplus} = v_n \cdots v_1$, $w_{\blacktriangleleft\ominus} = v'_n \cdots v'_1$. It holds that $xy \in L(M)$ for all $x \in \text{shuffle}(w_{\blacktriangleright\oplus}, w_{\blacktriangleleft\ominus})$ and $y \in \text{shuffle}(w_{\blacktriangleleft\oplus}, w_{\blacktriangleright\ominus})$.*

Proof. Since M satisfies the 1-limited restriction, exactly one symbol is always being read with a reading step. Therefore, for all i, only one of u_i, v_i, u'_i, v'_i is nonempty, and it contains one symbol. When M follows an accepting computation and a head of M jumps over a symbol with a $\oplus$-jumping step, such a symbol is read later with the other head of M with a $\ominus$-reading step. Since jumping steps can occur arbitrarily between reading steps and since they do not depend on the current state of M, it follows that every xy, where x is a shuffle of $w_{\blacktriangleright\oplus}$ and $w_{\blacktriangleleft\ominus}$ and y is a shuffle of $w_{\blacktriangleleft\oplus}$ and $w_{\blacktriangleright\ominus}$, has to belong to $L(M)$. □

Lemma 5.5.5. *There is no **1** jumping $5' \to 3'$ WK automaton M such that $L(M) = \{w_1 ccw_2 : w_1, w_2 \in \{a, b\}^*, \ |w_1 w_2|_a = |w_1 w_2|_b\}$.*

Proof. Basic idea. We follow the proof structure of Lemma 5.4.9. Considering any sufficiently large constant k, we show that M cannot process all symbols of $a^{3k} b^{3k} ccb^{3k} a^{3k}$ using only configurations that have their debt bounded by k.

Formal proof. By contradiction. Let $L = \{w_1 ccw_2 : w_1, w_2 \in \{a, b\}^*, \ |w_1 w_2|_a = |w_1 w_2|_b\}$, and let $M = (V, Q, q_0, F, \delta)$ be a **1** jumping $5' \to 3'$ WK automaton such that $L(M) = L$. Due to Lemma 5.4.8, there must exist a constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k. Consider any k for M such that $k \geq 2$. Let $w = a^{3k} b^{3k} ccb^{3k} a^{3k}$.

Consider restrictions on how M can accept w so that it does not also accept any $w' \notin L$. Due to Lemma 5.5.4, to ensure that both c's are always next to each other, some parts of $w_{\blacktriangleright\oplus}$, $w_{\blacktriangleright\ominus}$, $w_{\blacktriangleleft\oplus}$, $w_{\blacktriangleleft\ominus}$ must remain empty.

Consider cases where two or three parts remain empty. To ensure the proper position of c's, only one head can read or only $\oplus$-reading or only $\ominus$-reading steps can be used. Note

that the debt of an initial configuration of M is always 2 since at least cc has to be processed before an input can be successfully accepted. First, M cannot accept w with only one head because in this case jumping steps cannot be effectively used, the debt of the configuration of M reaches k after $k-2$ reading steps, and no further reading is possible. Second, the situation is similar for M using only $\oplus$-reading steps. Third, for M using only $\ominus$-reading steps, the heads can meet between a's and b's and process up to $7k+2$ symbols from w, but M is clearly still not able to process the whole w.

If only one part remains empty, the appropriate opposite part for the shuffle must contain both c's. Let us assume that $w_{\blacktriangleleft\ominus}$ remains empty. Consequently, $w_{\blacktriangleright\oplus}$ must contain at least $a^{3k}b^{3k}cc$. Consider possibilities how the $\blacktriangleright$-head can process $a^{3k}b^{3k}cc$ from w with $\oplus$-reading-steps. To process more than $k-2$ symbols a with the $\blacktriangleright$-head, both heads has to cooperate. Let us assume that M first reads $k-2$ times a with the $\blacktriangleleft$-head. Then, the $\blacktriangleleft$-head jumps to b's on the right-hand side of w, and the heads can start cooperate. The $\blacktriangleleft$-head reads b, the $\blacktriangleright$-head reads a, and this can be repeated $3k$ times. Now, the $\blacktriangleright$-head still has to process $b^{3k}cc$. Since there is the debt of $k-2$ symbols b created with the initial readings of the $\blacktriangleleft$-head, the $\blacktriangleright$-head can read $2k-4$ times b before the debt of the configuration of M reaches k. The $\blacktriangleright$-head still has to process $b^{k+4}cc$, but the debt cannot be compensated with the $\blacktriangleleft$-head any further. Consequently, the $\blacktriangleright$-head cannot process $a^{3k}b^{3k}cc$ from w with $\oplus$-reading-steps. The proof strategy and results are analogous for the other cases where $w_{\blacktriangleright\ominus}$, $w_{\blacktriangleleft\oplus}$, or $w_{\blacktriangleleft\ominus}$ remains empty.

Finally, we can see that M is not able to accept $w = a^{3k}b^{3k}ccb^{3k}a^{3k}$ when the debt of configurations of M is bounded by k. Since, for any k, $w \in L$, there is no constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k. But that is a contradiction with the assumption that there is a **1** jumping $5' \to 3'$ WK automaton M such that $L(M) = \{w_1ccw_2 : w_1, w_2 \in \{a,b\}^*,\ |w_1w_2|_a = |w_1w_2|_b\}$. □

Theorem 5.5.6. ***1 JWK*** $\subset$ ***JWK***.

Proof. This theorem follows directly from Example 5.5.3 and Lemma 5.5.5. □

Example 5.5.7. Consider the following **1** jumping $5' \to 3'$ WK automaton $M = (\{a,b\}, \{s,p\}, s, \{s\}, \delta)$ with the state transition function δ:

$$\delta(s,a,\varepsilon,\oplus) = \{p\}, \quad \delta(p,\varepsilon,b,\oplus) = \{s\},$$
$$\delta(s,a,\varepsilon,\ominus) = \{p\}, \quad \delta(p,\varepsilon,b,\ominus) = \{s\}.$$

It is not hard to see that the resulting behavior is similar to Example 5.3.1. The automaton now reads a's and b's with separate steps and uses one auxiliary state that is not final. Consequently, $L(M) = \{w \in \{a,b\}^* : |w|_a = |w|_b\}$.

Lemma 5.5.8. *For every linear grammar G, there is a **1** jumping $5' \to 3'$ WK automaton M such that $L(G) = L(M)$.*

Proof. By construction. Consider any linear grammar $G = (N,T,P,S)$. Every linear grammar has an equivalent grammar with rules in the form: (1) $S \to \varepsilon$, (2) $A \to aB$, (3) $A \to Ba$, (4) $A \to a$, where $A \in N$, $B \in (N - \{S\})$, and $a \in T$. Without loss of generality, assume that G satisfies this special form of rules and $q_f \notin (N \cup T)$. Define the **1** jumping $5' \to 3'$ WK automaton $M = (T, N \cup \{q_f\}, S, F, \delta)$, where F and δ are constructed in the following way:
(1) Set $F = \{q_f\}$. If $S \to \varepsilon \in P$, add S to F.

(2) For each $A \to aB \in P$, add B to $\delta(A, a, \varepsilon, \oplus)$.
(3) For each $A \to Ba \in P$, add B to $\delta(A, \varepsilon, a, \oplus)$.
(4) For each $A \to a \in P$, add q_f to $\delta(A, a, \varepsilon, \oplus)$.
Following the same reasoning as in Lemma 5.4.1, $L(M) = L(G)$. □

Theorem 5.5.9. ***LIN*** $\subset$ ***1 JWK***.

Proof. This theorem follows directly from Example 5.5.7 and Lemma 5.5.8. □

5.5.3 All-final Restriction

Lemma 5.5.10. *There is no* ***F*** *jumping* $5' \to 3'$ *WK automaton* M *such that* $L(M) = \{ca^n cb^n c : n \geq 0\} \cup \{\varepsilon\}$.

Proof. By contradiction. Assume that there is an **F** jumping $5' \to 3'$ WK automaton $M = (V, Q, q_0, F, \delta)$ such that $L(M) = \{ca^n cb^n c : n \geq 0\} \cup \{\varepsilon\}$. Since M satisfies the all-final restriction, all states are final. Therefore, if in the first nonempty reading step the ▶-head reads u and the ◀-head reads v, then uv or vu belongs to $L(M)$. Let $k_{\min} = \max\{|uv| : \delta(q, u, v, s) \neq \emptyset,\ q \in Q,\ u, v \in V^*,\ s \in \{\oplus, \ominus\}\}$. Consider any k such that $k > k_{\min}$. Let $w = ca^k cb^k c$. It is not hard to see that for any first nonempty reading step on w (which reads u and v) it must hold that $|uv|_c \leq 2$. However, for all $w' \in (L(M) - \{\varepsilon\})$ it holds that $|w'|_c = 3$. Therefore, if M accepts w, it also accepts $uv \notin L(M)$ or $vu \notin L(M)$. But that is a contradiction with the assumption that M exists. □

Theorem 5.5.11. ***F JWK*** $\subset$ ***JWK***.

Proof. This theorem follows directly from Theorem 5.4.2 and Lemma 5.5.10. □

Proposition 5.5.12. ***F JWK*** *and* ***LIN*** *are incomparable.*

Proof. **LIN** $\not\subseteq$ **F JWK** follows from Lemma 5.5.10. **F JWK** $\not\subseteq$ **LIN** follows from Example 5.3.1. Lastly, **F JWK** and **LIN** contain the language $\{a\}^*$. □

Lemma 5.5.13. *For every* $L \in$ ***F JWK*** *it holds that* $\varepsilon \in L$.

Proof. Consider any **F** jumping $5' \to 3'$ WK automaton $M = (V, Q, q_0, F, \delta)$. Since $Q = F$, q_0 is a final state and $(q_0, \oplus, \varepsilon, \varepsilon, \varepsilon) \curvearrowright (q_0, \ominus, \varepsilon, \varepsilon, \varepsilon)$ can be done with a $\oplus$-jumping step; thus, $\varepsilon \in L(M)$. □

Proposition 5.5.14. ***F JWK*** *and* ***FIN*** *are incomparable.*

Proof. **FIN** $\not\subseteq$ **F JWK** follows from Lemma 5.5.13. **F JWK** $\not\subseteq$ **FIN** follows from Example 5.3.1. Lastly, it is trivial to construct an **F** jumping $5' \to 3'$ WK automaton with two states that accepts the finite language $\{\varepsilon, a\}$. □

Theorem 5.5.15. ***FIN***$_{\varepsilon\text{-}inc}$ $\subset$ ***F JWK***.

Proof. By construction. Consider any alphabet V and $L = \{x_1, \ldots, x_n\} \in$ **FIN**$_{\varepsilon\text{-inc}}$ such that $x_i \in V^*$, $i = 1, \ldots, n$, $n \geq 1$. Define the **F** jumping $5' \to 3'$ WK automaton $M = (V, \{q_0, q_f\}, q_0, \{q_0, q_f\}, \delta)$, where δ is constructed in the following way: For each $x \in L$, set $\delta(q_0, x, \varepsilon, \oplus) = \{q_f\}$. It is clear that $L(M) = L$. Thus, **FIN**$_{\varepsilon\text{-inc}} \subseteq$ **F JWK**. **F JWK** $\not\subseteq$ **FIN**$_{\varepsilon\text{-inc}}$ follows from Example 5.3.1. □

Example 5.5.16. Consider the following **F** (in fact, even **N**) jumping $5' \to 3'$ WK automaton $M = (\{a, b, c\}, \{s\}, s, \{s\}, \delta)$ with the state transition function δ:

$$\delta(s, a, b, \oplus) = \{s\}, \quad \delta(s, a, b, \ominus) = \{s\},$$
$$\delta(s, cc, \varepsilon, \oplus) = \{s\}, \quad \delta(s, \varepsilon, cc, \oplus) = \{s\}.$$

This is a modification of Examples 5.3.1 and 5.5.3. The first two transitions ensure that M can accept any input containing the same number of a's and b's. The other two transitions ensure that the accepted inputs can also contain an arbitrary number of substrings cc. Therefore, $L(M) = \{w \in \{a, b, cc\}^* : |w|_a = |w|_b\}$.

Proposition 5.5.17. ***F JWK*** *and* ***1 JWK*** *are incomparable.*

Proof. First, **1 JWK** $\not\subseteq$ **F JWK** follows from Theorem 5.5.9 and Lemma 5.5.10. Second, let L be the language $L(M)$ from Example 5.5.16. The proof by contradiction from Lemma 5.5.5 can be modified in a straightforward way so that it shows that there is no **1** jumping $5' \to 3'$ WK automaton M such that $L(M) = L$. Therefore, **F JWK** $\not\subseteq$ **1 JWK**. Lastly, both families contain $\{a\}^*$. □

5.5.4 Stateless Restriction

Lemma 5.5.18. *There is no* ***N*** *jumping* $5' \to 3'$ *WK automaton* $M = (V, \{q_0\}, q_0, \{q_0\}, \delta)$ *such that* $L(M) \in$ ***FIN*** *and* $L(M) \neq \{\varepsilon\}$.

Proof. First, due to Lemma 5.5.13, $L(M)$ must always contain ε. Second, by contradiction, assume that there is a **N** jumping $5' \to 3'$ WK automaton M_2 such that $L(M_2) \in$ **FIN** and $L(M_2)$ contains a nonempty string. Since there is only one state, any $\oplus/\ominus$-reading step can be repeated arbitrarily many times. Therefore, if in the first nonempty reading step the ▶-head reads u and the ◀-head reads v, then $u^i v^i$ or $v^i u^i$ belongs to $L(M_2)$ for all $i \geq 1$. Thus, if M_2 accepts a nonempty string, $L(M_2) \notin$ **FIN**. But that is a contradiction with the assumption that M_2 exists. Consequently, if $L(M) \in$ **FIN**, $L(M) = \{\varepsilon\}$. □

Theorem 5.5.19. ***N JWK*** $\subset$ ***F JWK***.

Proof. From Lemma 5.5.1, **N JWK** $\subseteq$ **F JWK**. **F JWK** $\not\subseteq$ **N JWK** follows from Theorem 5.5.15 and Lemma 5.5.18. □

Proposition 5.5.20. ***N JWK*** *is incomparable with* ***LIN***, ***FIN***, *and* $\mathbf{FIN}_{\varepsilon\text{-inc}}$.

Proof. **LIN**, **FIN**, $\mathbf{FIN}_{\varepsilon\text{-inc}} \not\subseteq$ **N JWK** follows from Lemma 5.5.18. **N JWK** $\not\subseteq$ **LIN**, **FIN**, $\mathbf{FIN}_{\varepsilon\text{-inc}}$ follows from Example 5.3.1. Next, **N JWK** and **LIN** contain the language $\{a\}^*$. Finally, there is the sole language $\{\varepsilon\}$ that **N JWK** shares with **FIN** and $\mathbf{FIN}_{\varepsilon\text{-inc}}$. □

Proposition 5.5.21. ***N JWK*** *and* ***1 JWK*** *are incomparable.*

Proof. First, **1 JWK** $\not\subseteq$ **N JWK** follows from Theorem 5.5.9 and Lemma 5.5.18. Second, **N JWK** $\not\subseteq$ **1 JWK** follows from Example 5.5.16 and the proof of Proposition 5.5.17. Lastly, both families contain the language $\{a\}^*$. □

5.5.5 Combined Restrictions

Proposition 5.5.22. ***FS JWK*** $\subset$ ***F JWK***.

Proof. Let $L = \{cca^n cc : n \geq 0\} \cup \{\varepsilon\}$. It is trivial to construct an **F** jumping $5' \to 3'$ WK automaton that accepts L. However, there is no **FS** jumping $5' \to 3'$ WK automaton that accepts L. By contradiction. Assume that there is an **FS** jumping $5' \to 3'$ WK automaton M such that $L(M) = L$. Using the basic premise of Lemma 5.5.10, all c's has to be read with the first nonempty reading step. Nonetheless, a single head cannot read all c's in one step if they are arbitrarily far away from each other—a contradiction with the assumption that M exists. □

Theorem 5.5.23. ***FIN***$_{\varepsilon\text{-inc}}$ $\subset$ ***FS JWK***.

Proof. **FS JWK** $\not\subseteq$ **FIN**$_{\varepsilon\text{-inc}}$ follows from $\{a\}^* \in$ **FS JWK**. The rest of the proof is analogous to Theorem 5.5.15. □

Example 5.5.24. Consider the following **FS** jumping $5' \to 3'$ WK automaton $M = (\{a, b, c\}, \{s, p\}, s, \{s, p\}, \delta)$ with the state transition function δ:

$$\begin{aligned} \delta(s, a, \varepsilon, \oplus) &= \{p\}, \quad \delta(p, \varepsilon, b, \oplus) = \{s\}, \\ \delta(s, a, \varepsilon, \ominus) &= \{p\}, \quad \delta(p, \varepsilon, b, \ominus) = \{s\}, \\ \delta(s, cc, \varepsilon, \oplus) &= \{s\}, \quad \delta(s, \varepsilon, cc, \oplus) = \{s\}, \\ \delta(p, cc, \varepsilon, \oplus) &= \{p\}, \quad \delta(p, \varepsilon, cc, \oplus) = \{p\}. \end{aligned}$$

As a result, $L(M) = \{w \in \{a, b, cc\}^* : |w|_a = |w|_b \text{ or } |w|_a = |w|_b + 1\}$.
This automaton is just a combination of previous approaches from Examples 5.5.7 and 5.5.16. Note that $L(M)$ resembles the resulting language of Example 5.5.16.

Proposition 5.5.25. ***FS JWK*** *and* ***1 JWK*** *are incomparable.*

Proof. First, **1 JWK** $\not\subseteq$ **FS JWK** follows from the language in the proof of Proposition 5.5.22. Second, let L be the language $L(M)$ from Example 5.5.24. The proof by contradiction from Lemma 5.5.5 can be modified in a straightforward way so that it shows that there is no **1** jumping $5' \to 3'$ WK automaton M such that $L(M) = L$. Therefore, **FS JWK** $\not\subseteq$ **1 JWK**. Lastly, **FS JWK** and **1 JWK** contain the language $\{a\}^*$. □

Proposition 5.5.26. ***F1 JWK*** $\subset$ ***FS JWK***.

Proof. From Lemma 5.5.1, **F1 JWK** $\subseteq$ **FS JWK**. It is trivial to construct an **FS** jumping $5' \to 3'$ WK automaton that accepts $\{aa\}^*$. However, there cannot be an **F1** jumping $5' \to 3'$ WK automaton that accepts only even-length inputs. □

Proposition 5.5.27. ***F1 JWK*** *and* ***LIN*** *are incomparable.*

Proof. **LIN** $\not\subseteq$ **F1 JWK** follows from $\{aa\}^* \in$ **LIN**. Considering Example 5.5.24, there is an **F1** jumping $5' \to 3'$ WK automaton M such that $L(M) = \{w \in \{a, b\}^* : |w|_a = |w|_b \text{ or } |w|_a = |w|_b + 1\}$. Clearly, $L(M)$ is not a linear language. Lastly, **F1 JWK** and **LIN** contain the language $\{a\}^*$. □

Corollary 5.5.28. ***F1 JWK*** $\subset$ ***1 JWK***. □

Theorem 5.5.29. ***NS JWK*** $\subset$ ***REG***.

Proof. **NS JWK** $\subseteq$ **REG** can be proven by construction. We show that for any **NS** jumping $5' \to 3'$ WK automaton we can construct a finite automaton that accepts the same language. Consider any **NS** jumping $5' \to 3'$ WK automaton $M = (V, \{q_0\}, q_0, \{q_0\}, \delta)$. The following claims hold:

Claim 5.5.29.1. *Any* $w \in L(M)$ *can be expressed in the following special form* $w = x_1 y'_1 \cdots x_n y'_n x'_1 y_1 \cdots x'_m y_m$, *where* $x_i, y'_i, x'_j, y_j \in V^*$, *for all* $i = 1, \ldots, n$ *and* $j = 1, \ldots, m$, *for some* $n, m \geq 1$, *and for all* x_i, y'_i, x'_j, y_j *hold:*
(i) either $x_i = \varepsilon$ *or* $\delta(q_0, x_i, \varepsilon, \oplus) = \{q_0\}$,
(ii) either $y_j = \varepsilon$ *or* $\delta(q_0, \varepsilon, y_j, \oplus) = \{q_0\}$,
(iii) either $x'_j = \varepsilon$ *or* $\delta(q_0, x'_j, \varepsilon, \ominus) = \{q_0\}$,
(iv) either $y'_i = \varepsilon$ *or* $\delta(q_0, \varepsilon, y'_i, \ominus) = \{q_0\}$.

Proof. Due to the restrictions, parts (i), (ii), (iii), and (iv) cover all possible types of state transitions. The accepted input can be always divided into two parts, depending on the position where the heads of M meet each other during the processing of this input. The first part $x_1 y'_1 \cdots x_n y'_n$ is a combination of $\oplus$-readings with the $\blacktriangleright$-head and $\ominus$-readings with the $\blacktriangleleft$-head. Likewise, the second part $x'_1 y_1 \cdots x'_m y_m$ is a combination of $\ominus$-readings with the $\blacktriangleright$-head and $\oplus$-readings with the $\blacktriangleleft$-head. To get the uncertain reading order forced by the jumping steps, we also allow each part x_i, y'_i, x'_j, y_j to be empty. Therefore, all $w \in L(M)$ have to be able to satisfy this special form. ■

Claim 5.5.29.2. *Any* $w \in V^*$ *that can be expressed in the previous special form belongs to* $L(M)$.

Proof. Considering the restrictions, M has only one state, and only one head can read in a step. Therefore, if there is a possible reading step, it can be used arbitrarily many times. Furthermore, the possible reading steps can change only when the heads meet each other. Also, since each head reads separately, there cannot be any dependence between the first and second part of the input in the special form. Consequently, any $w \in V^*$ that can be expressed in the form from Claim 1 has to belong to $L(M)$. ■

Considering both claims, it is easy to construct a finite automaton that accepts all inputs of this special form. **REG** $\not\subseteq$ **NS JWK** follows from Lemma 5.5.18. □

Proposition 5.5.30. ***N1 JWK*** $\subset$ ***NS JWK***.

Proof. This proof is analogous to that of Proposition 5.5.26. □

Proposition 5.5.31. *The following relations hold:*
(i) ***NS JWK*** $\subset$ ***N JWK***;
(ii) ***NS JWK*** $\subset$ ***FS JWK***;
(iii) ***N1 JWK*** $\subset$ ***F1 JWK***.

Proof. Examples 5.3.1 and 5.5.24 and Proposition 5.5.27 show that **N JWK**, **FS JWK**, and **F1 JWK** contain some non-regular languages. Considering Lemma 5.5.1 and Theorem 5.5.29, all three proposed relations directly follow. □

Proposition 5.5.32. ***FS JWK*** *and* ***N JWK*** *are incomparable.*

Proof. First, **FS JWK** $\not\subseteq$ **N JWK** follows from Lemma 5.5.18 and Theorem 5.5.23. Second, let $L = \{a^n b^n : n \geq 0\}$. It is trivial to construct an **N** jumping $5' \to 3'$ WK automaton that accepts L. However, there is no **FS** jumping $5' \to 3'$ WK automaton that accepts L. By contradiction. Assume that there is an **FS** jumping $5' \to 3'$ WK automaton $M = (V, Q, q_0, F, \delta)$ such that $L(M) = L$. Due to the restrictions, if a head of M reads u in a step, it must hold that $|u|_a = |u|_b$. Otherwise, there would be $w' \in L(M)$ such that $|w'|_a \neq |w'|_b$. Let $k_{\min} = \max\{|v_1 v_2| : \delta(q, v_1, v_2, s) \neq \emptyset,\ q \in Q,\ v_1, v_2 \in V^*,\ s \in \{\oplus, \ominus\}\}$. Consider any k such that $k > k_{\min}$. Let $w = a^{2k} b^{2k}$. Clearly, when M processes w, each head can read u such that $|u|_a = |u|_b$ no more than once. However, these balanced steps can therefore process only less than $2k$ symbols. Consequently, if M accepts w, it also accepts some $w' \notin L$—a contradiction with the assumption that M exists. Therefore, **N JWK** $\not\subseteq$ **FS JWK**. Lastly, **FS JWK** and **N JWK** contain the language $\{a\}^*$. □

Proposition 5.5.33. ***F1 JWK*** *and* ***NS JWK*** *are incomparable.*

Proof. First, **F1 JWK** $\not\subseteq$ **NS JWK** follows from Lemma 5.5.18 and $\{\varepsilon, a\} \in$ **F1 JWK**. Second, **NS JWK** $\not\subseteq$ **F1 JWK** follows from $\{aa\}^* \in$ **NS JWK**. Lastly, both families contain the language $\{a\}^*$. □

Proposition 5.5.34. ***REG*** *is incomparable with* ***F JWK***, ***N JWK***, ***FS JWK***, *and* ***F1 JWK***.

Proof. First, Examples 5.3.1 and 5.5.24 and Proposition 5.5.27 show that **F JWK**, **N JWK**, **FS JWK**, and **F1 JWK** contain some non-regular languages. Second, let $L = \{ca^n cb^m c : n, m \geq 0\} \cup \{\varepsilon\}$. L is clearly a regular language. Considering the proof of Lemma 5.5.10 and the previous results, we can easily see that **F JWK**, **N JWK**, **FS JWK**, and **F1 JWK** cannot contain L. Lastly, all families contain the language $\{a\}^*$. □

Proposition 5.5.35. ***FIN*** *is incomparable with* ***FS JWK***, ***F1 JWK***, ***NS JWK***, *and* ***N1 JWK***.

Proof. Considering previous results. First, **FS JWK**, **F1 JWK**, **NS JWK**, and **N1 JWK** cannot contain $\emptyset$. Second, **FS JWK**, **F1 JWK**, **NS JWK**, and **N1 JWK** contain $\{a\}^*$. Lastly, all families contain $\{\varepsilon\}$. □

Proposition 5.5.36. ***FIN***$_{\varepsilon\text{-inc}}$ *is incomparable with* ***F1 JWK***, ***NS JWK***, *and* ***N1 JWK***.

Proof. Considering previous results. First, **F1 JWK**, **NS JWK**, and **N1 JWK** cannot contain $\{\varepsilon, aa\}$. Second, **F1 JWK**, **NS JWK**, and **N1 JWK** contain $\{a\}^*$. Lastly, all families contain $\{\varepsilon\}$. □

All the obtained results comparing the accepting power of different variants of jumping $5' \to 3'$ WK automata are summarized in Figure 5.1.

5.6 Concluding Remarks

The results clearly show that, with the addition of the jumping mechanism into the model, the accepting power has been increased above sensing $5' \to 3'$ WK automata. The model is now able to accept some nonlinear and even some non-context-free languages. On the other hand, the jumping movement of the heads is restricted compared to jumping finite

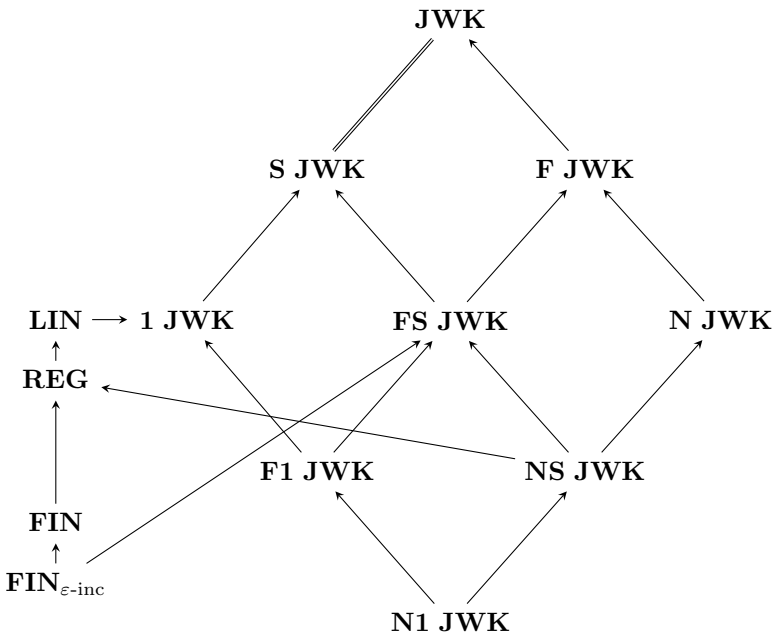

Figure 5.1: A hierarchy of language families closely related to the unrestricted and restricted variants of jumping $5' \to 3'$ WK automata is shown. If there is a double line between families X and Y, then $X = Y$. If there is an arrow from family X to family Y, then $X \subset Y$. Furthermore, if there is no path (following the arrows and double lines) between families X and Y, then X and Y are incomparable.

automata, and this limits its capabilities to accept languages that require a more sophisticated discontinuous information processing. Considering the comparison with full-reading sensing $5' \to 3'$ WK automata, the results are not yet clear and further research is required. However, we know that there are some languages, like $\{a^n b^n c^n : n \geq 0\}$, that cannot be accepted by jumping $5' \to 3'$ WK automata and that are accepted by full-reading sensing $5' \to 3'$ WK automata (see [60, 61, 63, 65]).

If we compare the hierarchies of language families related to the restricted variants of jumping $5' \to 3'$ WK automata and sensing $5' \to 3'$ WK automata (see [63, 65, 70]), there are several noticeable remarks. Most importantly, the 1-limited restriction (**1**) has a negative impact on the accepting power, which is usually not the case in sensing $5' \to 3'$ WK automata. Secondly, when several restrictions are combined together, the hierarchy structure resembles the alternative structure of sensing $5' \to 3'$ WK automata without the sensing distance from [70]. Lastly, almost all restricted variants, with the exception of **NS** and **N1**, are still able to accept some nonlinear languages, which cannot be accepted by any variants of sensing $5' \to 3'$ WK automata.

The reader may notice that the $\ominus$-jumping can be used only in situations where it is forced by the current configuration. Jumping finite automata usually immediately erase symbols from the configuration and do not use the auxiliary symbol #. It is therefore a question for future research whether we can safely remove this part from the model and keep the accepting power intact.

Part III

New Results on CD Grammar Systems

And now for something completely different.
—Monty Python's Flying Circus

Chapter 6

General CD Grammar Systems: Normal Forms

This chapter introduces new normal forms of general CD grammar systems fitting for a parallel rewriting process. The content of this chapter is composed of results that were first introduced in a short abstract at the conference AFL 2017 and later published in Journal of Automata, Languages and Combinatorics (see [35]); all written jointly with Zbyněk Křivka and Alexander Meduna.

In terms of preliminaries, the reader should be familiar with the definitions of general notions (see Section 2.1), grammars (see Section 2.2.1), Kuroda normal form (see Section 2.2.4), homogeneous restrictions (see Section 2.2.5), and general CD grammar systems (see Section 2.4).

6.1 Introduction

The present chapter, which assumes a familiarity with formal language theory (see [49, 91]), concerns grammar systems (see [8]). It concentrates its attention on two-component CD grammar systems working under the $*$ and t modes. Recall that under the former mode the context-free versions of these systems obviously generate only the family of context-free languages. More surprisingly, under the latter mode they are no more powerful than ordinary context-free grammars either. To increase their power, the present work uses general CD grammar systems, whose components are general grammars, that are computationally complete—that is, they characterize the family of recursively enumerable languages. Most importantly, however, we explain how to turn arbitrary general grammars into equivalent two-component general CD grammar systems of very reduced and simplified forms.

To give an insight into this study in a greater detail, take any general grammar G. This chapter demonstrates two types of transformations that turn G into a two-component general CD grammar system with one context-free component and one non-context-free component. For brevity, in this introductory section, Γ_1 and Γ_2 denote the systems resulting from the first type of transformations and the second type of transformations, respectively. Γ_1 has its non-context-free component containing the rules $11 \to 00$ and $0000 \to \varepsilon$, while Γ_2 has its non-context-free component containing the rules $11 \to 00$ and $0000 \to 2222$, where 0, 1, and 2 are new nonterminals. The chapter proves that working under the $*$ and t modes, Γ_1 and Γ_2 are equivalent to G. Thus, more generally speaking, general CD grammar systems of these two forms are computationally complete—that is, they characterize the

family of recursively enumerable languages. Apart from the computational completeness, it is worth mentioning the following other useful properties, (i) through (v), which make Γ_1 and Γ_2 simple and easy to apply in theory as well as in practice.

(i) Most importantly, observe that Γ_1 and Γ_2 utilize a very reduced number of non-context-free rules. One of their components is always purely context-free, and the other has only two non-context-free rules. Of course, computational completeness resulting from such strongly reduced versions of general CD grammar systems is more than highly appreciated from both a theoretical and practical standpoint.

(ii) Consider Γ_1. The paper demonstrates that working under the t mode, during every generation of a sentence, Γ_1 changes its components no more than once. Furthermore, if the system simulates the use of at least one non-context-free rule from the original grammar, it changes its components precisely once.

(iii) From a general viewpoint, taking a closer look at language-generating rewriting systems, we intuitively see that some of them generate sentences of the same language in a more similar way than others. Formal language theory has formalized this generative phenomenon in terms of close derivation simulations (see Chapter 6 in [55] and [56]). To give an insight into this formalization, consider grammatical models X and Y. Let $\Rightarrow$ denote a derivation step, and let $\Rightarrow^m$ denote m consecutive derivation steps. If there is a constant k such that for every derivation of the form

$$x_0 \Rightarrow x_1 \Rightarrow \cdots \Rightarrow x_n$$

in X, where x_0 is its start symbol, there is a derivation of the form

$$x_0 \Rightarrow^{k_1} x_1 \Rightarrow^{k_2} \cdots \Rightarrow^{k_n} x_n$$

in Y, where $k_i \leq k$ for each $1 \leq i \leq n$, we say that Y closely simulates X. In this sense, the chapter demonstrates that under the $*$ mode Γ_1 and Γ_2 in many cases closely simulate G. This property also makes these transformations distinct from some well-known transformations generating grammatical models with a reduced number of non-context-free rules (e.g., Geffert normal forms [17, 19]), since they usually require a very strict derivation flow for the resulting model.

(iv) The specific form of the rules in Γ_1 and Γ_2 makes it possible to utilize parallelization through the whole sentence generation process. In essence, it is possible to use multi-derivations that, during a derivation step, rewrite the sentential form at several positions at once, not just at a single position. This property also holds for uniform derivations that always rewrite at all possible positions at once.

(v) Before sketching the final property, we recall that a grammatical rule of the form $x \to y$, where x and y are strings, is homogeneous if x is formed by a string of identical symbols (see Section 2.2.5 or [51]). A homogeneous rule $x \to y$ is evenly homogeneous if y is also formed by a string of identical symbols and $|x| = |y|$. In a CD grammar system, a component is homogeneous if all its rules are homogeneous, and it is evenly homogeneous if all its rules are evenly homogeneous. A CD grammar system is rule-homogeneous if all its components are homogeneous. As obvious, any CD grammar system with context-free components is rule-homogeneous. Observe that Γ_1 and Γ_2 are both rule-homogeneous CD grammar systems with one context-free component

and one homogeneous component. In fact, in Γ_2, the second component is evenly homogeneous.

The rest of this chapter is organized as follows. Section 6.2 explores properties of general CD grammar systems. Section 6.3 introduces all fundamental techniques of the transformations on input grammars that satisfy Kuroda normal form. Sections 6.4 generalizes the previous techniques for arbitrary general grammars and presents the remaining results. Section 6.5 closes the study by pointing out some remarks and suggestions for further investigation.

6.2 On General CD Grammar Systems

The definition of general CD grammar systems (see Section 2.4) can be easily modified so that the components are sets of rules of any type. Recall that, for CD grammar systems having regular, linear, context-sensitive, or general components, their generative power often does not change with the number of components (see [8, 78]), i.e., they still generate the families of regular, linear, context-sensitive, or recursively enumerable languages, respectively. Nonetheless, different results have been obtained by studying some other non-classical components—e.g., permitting, left-forbidding, and random context components (see [9, 20, 30])—where the number of components affects the resulting generative power.

It is clear that if we require computational completeness, we need components that use stronger mechanisms than basic context-free rules. In general, components with homogeneous rules have a similar effect as general components—a single homogeneous component can define **RE** by itself (see [51]). The same, however, does not hold for components with evenly homogeneous rules, which can define only sets of single symbols on their own. Therefore, one may wonder, what properties we can get if we combine together several relatively simple components of different types.

The rest of this chapter studies two-component general CD grammar systems where the first component is always purely context-free and the second component contains either homogeneous or evenly homogeneous rules. Furthermore, we limit the non-context-free component so it contains only two rules.

6.3 Transformations from Kuroda Normal Form

In order to simplify the reasoning for underlying proofs, this section assumes that all input grammars satisfy Kuroda normal form (see Section 2.2.4). Nonetheless, in the following section, we show that Kuroda normal form is not necessary and that we can use similar techniques to convert any general grammar into a two-component general CD grammar system that also satisfies similar properties.

First, let us start with the most straightforward variant of a two-component general CD grammar system that works in the $*$ mode and has the second component homogeneous. The following proof will also serve as a framework for later proofs since the majority of the reasoning can be shared throughout the variants.

Theorem 6.3.1. *Let $G = (N, T, P, S)$ be a grammar in Kuroda normal form. Then, there exists a two-component general CD grammar system $\Gamma = (N', T, H, I, S)$ such that H is context-free, $I = \{11 \to 00,\ 0000 \to \varepsilon\}$, and $L_*(\Gamma) = L(G)$.*

Proof.
Construction.
Let $G = (N, T, P, S)$. Without any loss of generality, assume that $(N \cup T) \cap \{0, 1\} = \emptyset$. For $m = 2 + \text{card}(\text{NonContextFree}(P))$, define an injection g from $\text{NonContextFree}(P)$ to $(\{01\}^+\{00\}\{01\}^+ \cap \{01, 00\}^m)$. From G, we construct the two-component general CD grammar system $\Gamma = (N', T, H, I, S)$, where $N' = N \cup \{0, 1\}$, $I = \{11 \to 00,\ 0000 \to \varepsilon\}$, and H is defined as follows:

(I) For every $AB \to CD \in P$ where $A, B, C, D \in N$,
add $A \to CDg(AB \to CD)$ and $B \to \text{rev}(g(AB \to CD))$ to H.

(II) For every $A \to x \in P$ where $A \in N$ and $x \in (\{\varepsilon\} \cup T \cup N^2)$, add $A \to x$ to H.

The construction of Γ is completed.

Basic idea.

(a) The rules (I) and the second component I simulate the derivation steps made by $\text{NonContextFree}(P)$ in G. That is, $xABy \Rightarrow xCDy$ according to $AB \to CD \in P$ in G, where $x, y \in (N \cup T)^*$, is simulated in Γ as

$$\begin{aligned} xABy &\Rightarrow_H xCDg(AB \to CD)By \\ &\Rightarrow_H xCDg(AB \to CD)\,\text{rev}(g(AB \to CD))y \\ &\Rightarrow_I^{2m-1} xCDy. \end{aligned}$$

Γ makes the $(2m-1)$-step derivation $xCDg(AB \to CD)\,\text{rev}(g(AB \to CD))y \Rightarrow_I^{2m-1} xCDy$ by using only the rules $11 \to 00$ and $0000 \to \varepsilon$. During this derivation, the string between D and y always contains exactly one occurrence of consecutive identical symbols that can be rewritten, so this derivation actually verifies that the simulation of $xABy \Rightarrow xCDy$ is made properly.

(b) The rules (II) simulate the use of $\text{ContextFree}(P)$ in G.

The reader may notice that the simulation of non-context-free rules resembles similar techniques used in general grammars (see [79, 16, 17, 18, 19, 51]). However, this is traditionally done either by using several types of matching parentheses (see [79, 16]), which is not a suitable form for homogeneous rules, or it requires a significant non-local change in the generation flow of the original grammar (see [17, 19, 51]), which denies the close derivation simulations.

Formal proof.
We prove $L_*(\Gamma) = L(G)$. It was already proven in part B of the proof of Theorem 1 in [16] and in the proof of Theorem 1 in [79] that the rules of the form $XY \to w$, where $X, Y \in N$, $w \in (N \cup T)^*$, can be replaced with $X \to wL_i$, $Y \to R_i$, $L_iR_i \to \varepsilon$, where L_i, R_i are new unique nonterminals for each rule. Thus, we only need to prove that our encoding with injection g simulates the same behavior and that it works in two-component general CD grammar systems.

First, we establish the terminology that we will use throughout this proof: Any consecutive sequence of nonterminals 0 and 1 is referred to as *verification code*. We say that a sequence is *rewritable* if there is a rule in Γ that can be used on the sequence. We recognize three distinct types of verification codes in the sentential form:

- *unconnected verification code* – This is the initial sequence from the rules (I). Considering some rule $AB \to CD \in P$, it is either $g(AB \to CD)$ or $\text{rev}(g(AB \to CD))$. In more detail, we identify $g(AB \to CD)$ as a *left unconnected verification code* (the

code of the form $\{01\}^+\{00\}\{01\}^+$) and $\text{rev}(g(AB \to CD))$ as a *right unconnected verification code* (the code of the form $\{10\}^+\{00\}\{10\}^+$).

- *connected verification code* – This sequence is established when some left and right unconnected verification codes merge together in the sentential form, and we identify it as connected until it has the form $\{01\}\{0,1\}^*\{10\}$.
- *leftover* – The remaining sequence $\{0000\}$.

Now, we show that our encoding properly simulates $L_iR_i \to \varepsilon$. First, we establish a claim on how a connected verification code can be rewritten.

Claim 6.3.1.1. *Let $AB \to CD \in P$, where $A, B, C, D \in N$. A (standalone) connected verification code can be reduced to a leftover if and only if it was initially established as $g(AB \to CD)\,\text{rev}(g(AB \to CD))$.*

Proof. Let $AB \to CD, EF \to UV \in P$, where $A, B, C, D, E, F, U, V \in N$, such that $g(AB \to CD) = (01)^k00(01)^l$ and $g(EF \to UV) = (01)^p00(01)^q$, where $k, l, p, q \geq 1$, $k + l + 1 = m$, $p + q + 1 = m$, $k \neq p$. There are two distinct cases how a connected verification code can be initially established: either $g(AB \to CD)rev(g(AB \to CD))$ or $g(AB \to CD)rev(g(EF \to UV))$. (There are four ways how to pair the unconnected verification codes but only two distinct cases since the rules can be swapped.)

The first case creates the sequence $(01)^k00(01)^l(10)^l00(10)^k$. Initially, the rules $11 \to 00$ and $0000 \to \varepsilon$ are used l times to erase $(01)^l(10)^l$. Then, the rule $0000 \to \varepsilon$ has to be used. Next, both rules are used $k - 1$ times again until only the sequence 0110 remains. And finally, the rule $11 \to 00$ creates the leftover.

In the second case, the sequence is $(01)^k00(01)^l(10)^q00(10)^p$. Assume that $l > q$; the result is analogous for the opposite situation. The rules $11 \to 00$ and $0000 \to \varepsilon$ can be used q times, and it leads to the sequence $(01)^k00(01)^{l-q}00(10)^p$. Nonetheless, this sequence cannot be rewritten any further.

In both cases, the derivation steps cannot be done in any other way. Thus, it is clear that a connected verification code can be reduced to a leftover if and only if it is established as $g(AB \to CD)\,\text{rev}(g(AB \to CD))$ for some $AB \to CD \in P$. ■

Next, we demonstrate that there is no sentential form in which a verification code could be rewritten in some unintended way.

Claim 6.3.1.2. *In any reachable sentential form, a verification code can be rewritten only if it can be identified as either a connected verification code or a sequence of 0's containing a leftover as its substring.*

Proof. Verification codes can be rewritten only with the rules $11 \to 00$ and $0000 \to \varepsilon$. Any verification code generated into the sentential form is initially in the form $\{01\}^+\{00\}\{01\}^+$ or $\{10\}^+\{00\}\{10\}^+$. Clearly, no rule can rewrite this code as long as it remains alone. When the left and right unconnected verification codes are joined together, a connected verification code is established where the rewriting can occur. In contrast, observe that unconnected verification codes joined in different ways do not establish any rewritable sequence.

Considering the proof of Claim 6.3.1.1, a connected verification code is always in the form $\{01\}\{0,1\}^*\{10\}$ until it is reduced to a leftover. Observe that if this form is joined together with other connected or unconnected verification codes, it does not establish any new rewritable sequence.

Lastly, the leftover 0000 is clearly rewritable on its own, but it can be also merged with other verification codes and that can create an even longer rewritable sequence of 0's. Nonetheless, observe that only the rule $0000 \to \varepsilon$ can be applied on this sequence, which erases precisely the leftover. Thus, this cannot affect the form of the other verification codes.

The above description covers all obtainable forms of verification codes, and only the connected verification code and the sequence of 0's containing a leftover as its substring can be rewritten. Thus, Claim 6.3.1.2 holds. ■

From Claims 6.3.1.1 and 6.3.1.2, it is obvious that the encoding successfully simulates the unique nonterminals L_i, R_i and the erasing rules $L_iR_i \to \varepsilon$.

Next, we prove $L(G) \subseteq L_*(\Gamma)$; more precisely, by induction on the number of derivation steps, we demonstrate Claim 6.3.1.3.

Claim 6.3.1.3. *For every $w \in (N \cup T)^*$ and $i \geq 0$, $S \Rightarrow^i w$ in G implies $S \Rightarrow^*_{k_1} w_1 \Rightarrow^*_{k_2} \cdots \Rightarrow^*_{k_l} w_l = w$, $l \geq 1$, $k_j \in \{H, I\}$, $1 \leq j \leq l$, in Γ.*

Proof. Basis: Let $i = 0$. Then, $w = S$. Clearly, $S \Rightarrow^*_H S$.
Induction hypothesis: Assume that the implication of Claim 6.3.1.3 holds for every $i \leq o$, where o is a non-negative integer.
Induction step: Consider any derivation of the form $S \Rightarrow^{o+1} \beta$ in G, where $\beta \in (N \cup T)^*$. Express $S \Rightarrow^{o+1} \beta$ as $S \Rightarrow^o \alpha \Rightarrow \beta$, where $\alpha \in (N \cup T)^*$. By the induction hypothesis, $S \Rightarrow^*_{k_1} w_1 \Rightarrow^*_{k_2} \cdots \Rightarrow^*_{k_l} w_l = \alpha$, $l \geq 1$, $k_j \in \{H, I\}$, $1 \leq j \leq l$, in Γ. There are the following two possibilities how G can make $\alpha \Rightarrow \beta$:

(1) Let $AB \to CD \in P$, $\alpha = xABy$, $\beta = xCDy$, $x, y \in (N \cup T)^*$, $A, B, C, D \in N$. According to (a) in the basic idea and from Claims 6.3.1.1 and 6.3.1.2,

$$\begin{aligned} xABy &\Rightarrow_H xCDg(AB \to CD)By \\ &\Rightarrow_H xCDg(AB \to CD)\,\mathrm{rev}(g(AB \to CD))y \\ &\Rightarrow^{2m-1}_I xCDy \end{aligned}$$

in Γ. Consequently, $S \Rightarrow^*_{k_1} w_1 \Rightarrow^*_{k_2} \cdots \Rightarrow^*_{k_{l'}} w_{l'} = xCDy = \beta$, $l' \geq 1$, $k_j \in \{H, I\}$, $1 \leq j \leq l'$, in Γ.

(2) Let $A \to z \in P$, $\alpha = xAy$, $\beta = xzy$, $x, y \in (N \cup T)^*$, $A \in N$, $z \in (\{\varepsilon\} \cup T \cup N^2)$. From (b) in the basic idea, $xAy \Rightarrow_H xzy$ in Γ. Consequently, $S \Rightarrow^*_{k_1} w_1 \Rightarrow^*_{k_2} \cdots \Rightarrow^*_{k_{l'}} w_{l'} = xzy = \beta$, $l' \geq 1$, $k_j \in \{H, I\}$, $1 \leq j \leq l'$, in Γ.

The induction step is completed, so Claim 6.3.1.3 holds. ■

Lastly, we prove $L_*(\Gamma) \subseteq L(G)$. We show that, for any $y \in L_*(\Gamma)$, there is a sequence of derivation steps in Γ that precisely follows the intended order from the basic idea, so $y \in L(G)$.

Claim 6.3.1.4. *Any successful derivation sequence generating $y \in L_*(\Gamma)$ in Γ can be reordered so it satisfies the form*

$$\begin{aligned} S = v_{0_3} &\Rightarrow^*_H v_{1_0} \Rightarrow_H v_{1_1} \Rightarrow_H v_{1_2} \Rightarrow^{2m-1}_I v_{1_3} \\ &\Rightarrow^*_H v_{2_0} \Rightarrow_H v_{2_1} \Rightarrow_H v_{2_2} \Rightarrow^{2m-1}_I v_{2_3} \\ &\vdots \\ &\Rightarrow^*_H v_{k_0} \Rightarrow_H v_{k_1} \Rightarrow_H v_{k_2} \Rightarrow^{2m-1}_I v_{k_3} \Rightarrow^*_H v_{(k+1)_0} = y, \end{aligned}$$

where for $i = 0, 1, \ldots, k$ *in* $v_{i_3} \Rightarrow_H^* v_{(i+1)_0}$ *every sentential form is over* $(N \cup T)^*$*; and for* $j = 1, \ldots, k$ *the sentential forms in the derivation* $v_{j_0} \Rightarrow_H v_{j_1} \Rightarrow_H v_{j_2} \Rightarrow_I^{2m-1} v_{j_3}$ *have the structure:*

$$
\begin{aligned}
&v_{j_0} = u_j A_j B_j w_j, \\
&v_{j_1} = u_j C_j D_j g(A_j B_j \to C_j D_j) B_j w_j, \\
&v_{j_2} = u_j C_j D_j g(A_j B_j \to C_j D_j) \operatorname{rev}(g(A_j B_j \to C_j D_j)) w_j, \\
&v_{j_3} = u_j C_j D_j w_j, \\
&\text{for some } A_j B_j \to C_j D_j \in P,\ u_j, w_j \in (N \cup T)^*,\ A_j, B_j, C_j, D_j \in N.
\end{aligned}
$$

Proof. First, all nonterminals in N can be rewritten only with the context-free rules of the component H. This implies that it does not matter in which order we rewrite them in the sentential form. Second, consider rules $A \to CDg(AB \to CD), B \to \operatorname{rev}(g(AB \to CD)) \in H$, where $A, B, C, D \in N$, $AB \to CD \in P$. Claims 6.3.1.1 and 6.3.1.2 show that only the verification code $g(AB \to CD) \operatorname{rev}(g(AB \to CD))$ can be successfully erased. It follows that we can always establish some order of derivations in which the sequence $v_{j_0} \Rightarrow_H v_{j_1} \Rightarrow_H v_{j_2} \Rightarrow_I^{2m-1} v_{j_3}$ holds for each simulated non-context-free rule. ■

From the reordered derivations of Claim 6.3.1.4 in Γ and from (I) and (II), we see that $v_{0_3} \Rightarrow^* v_{(k+1)_0}$ in G. Therefore, $y \in L_*(\Gamma)$ implies $y \in L(G)$. Thus, $L_*(\Gamma) \subseteq L(G)$.
As $L(G) \subseteq L_*(\Gamma)$ and $L_*(\Gamma) \subseteq L(G)$, $L_*(\Gamma) = L(G)$. Thus, Theorem 6.3.1 holds. □

Corollary 6.3.2. *The resulting two-component general CD grammar system* Γ *from the proof of Theorem 6.3.1 closely simulates the original grammar* G*.*

Proof. For any resulting Γ, we can find a bounded constant k such that for every possible derivation $u \Rightarrow v$ in G there is a k'-step derivation in Γ that gives the same result and $k' \leq k$. Furthermore, for a given Γ, we can easily determine the minimal possible k.

Consider the proof of Claim 6.3.1.3 and the mentioned possibilities how G can make $\alpha \Rightarrow \beta$. Any context-free rule is simulated in one derivation step. The non-context-free rules require two initial derivation steps and the rewriting of the verification code. The length of the rewriting depends on the size of m, and it takes $2m - 1$ steps to complete. The minimal possible k for a given Γ is therefore $2m + 1$. □

Next, we consider a two-component general CD grammar system with the same structure but working in the t mode.

Theorem 6.3.3. *Let* $G = (N, T, P, S)$ *be a grammar in Kuroda normal form. Then, there exists a two-component general CD grammar system* $\Gamma = (N', T, H, I, S)$ *such that* H *is context-free,* $I = \{11 \to 00,\ 0000 \to \varepsilon\}$*, and* $L_t(\Gamma) = L(G)$*.*

Proof.
Construction. The process of construction remains identical to Theorem 6.3.1. For a grammar $G = (N, T, P, S)$, $m = 2+\text{card}(\text{NonContextFree}(P))$, and injection g, we construct the two-component general CD grammar system $\Gamma = (N', T, H, I, S)$, where $N' = N \cup \{0, 1\}$, and H and I contain the rules as described in Theorem 6.3.1.

Basic idea.
Recall that, during the generation of a sentence, a CD grammar system working in the t mode switches its components only if the process is not finished and there are no possible

derivations with the previous component. Consider the general behavior of Γ. It starts the generation with S. For the first derivation, applicable rules can be found only in H, so this component has to be used. However, H also contains all rules simulating the original rules of G. Consequently, the first derivation in the t mode has to simulate all rules in G and cannot rewrite any generated verification codes. Nonetheless, we prove that the verification codes can be successfully erased afterwards for all simulated non-context-free rules at once.

Formal proof.
We prove $L_t(\Gamma) = L(G)$. First, let us prove the statement introduced above. For convenience, consider the homomorphism $\varphi : (N' \cup T)^* \to (N \cup T)^*$ where $\varphi(a) = a$ and $\varphi(b) = \varepsilon$, for all $a \in (N \cup T)$ and $b \in \{0, 1\}$.

Claim 6.3.3.1. *For every $u \in (N \cup T)^*$ and $i \geq 0$, $S \Rightarrow^i u$ in G implies $S \Rightarrow_H^* w \Rightarrow_I^t u$ in Γ, where $w \in (N' \cup T)^*$ and $\varphi(w) = u$. Furthermore, w satisfies the form $w = p_1 q_1 \cdots p_n q_n$, where $n \geq 1$, $p_j \in (N \cup T)^*$, $q_j \in \{0, 1\}^*$, $1 \leq j \leq n$, and every q_j represents a verification code that can be successfully erased on its own.*

Proof. Basis: Let $i = 0$. Then, $u = S$. Clearly, $S \Rightarrow_H^0 S \Rightarrow_I^t S$, and the required form also holds.
Induction hypothesis: Assume that Claim 6.3.3.1 holds for every $i \leq o$, where o is a non-negative integer.
Induction step: Consider any derivation of the form $S \Rightarrow^{o+1} \beta$ in G, where $\beta \in (N \cup T)^*$. Express $S \Rightarrow^{o+1} \beta$ as $S \Rightarrow^o \alpha \Rightarrow \beta$, where $\alpha \in (N \cup T)^*$. By the induction hypothesis, $S \Rightarrow_H^* w \Rightarrow_I^t \alpha$, where $\varphi(w) = \alpha$, in Γ. There are the following two possibilities how G can make $\alpha \Rightarrow \beta$:

(1) Let $AB \to CD \in P$, $\alpha = xABy$, $\beta = xCDy$, $x, y \in (N \cup T)^*$, $A, B, C, D \in N$. Consider w in the required form. Let $w = p_1 q_1 \cdots p_k A q_k B p_{k+1} q_{k+1} \cdots p_n q_n$, where $n \geq 1$, $1 \leq k \leq n$, $p_j \in (N \cup T)^*$, $q_j \in \{0, 1\}^*$, $1 \leq j \leq n$, and also $p_1 \cdots p_k = x$ and $p_{k+1} \cdots p_n = y$. Then,

$$
\begin{aligned}
& w = p_1 q_1 \cdots p_k A q_k B p_{k+1} q_{k+1} \cdots p_n q_n \\
& \Rightarrow_H p_1 q_1 \cdots p_k C D g(AB \to CD) q_k B p_{k+1} q_{k+1} \cdots p_n q_n \\
& \Rightarrow_H p_1 q_1 \cdots p_k C D g(AB \to CD) q_k \operatorname{rev}(g(AB \to CD)) p_{k+1} q_{k+1} \cdots p_n q_n \\
& = w'
\end{aligned}
$$

in Γ, and there are two possible situations regarding these steps:

(a) If $q_k = \varepsilon$, the steps add a new connected verification code. By Claims 6.3.1.1 and 6.3.1.2, such a code can be successfully erased on its own, so the required form holds. Consequently, $S \Rightarrow_H^* w' \Rightarrow_I^t \beta$ in Γ.

(b) If $q_k \neq \varepsilon$, the steps prolong some existing verification code. However, since q_k has to be erasable on its own, observe that this creates a properly nested structure that is also erasable on its own, so the required form holds. Consequently, $S \Rightarrow_H^* w' \Rightarrow_I^t \beta$ in Γ.

(2) Let $A \to z \in P$, $\alpha = xAy$, $\beta = xzy$, $x, y \in (N \cup T)^*$, $A \in N$, $z \in (\{\varepsilon\} \cup T \cup N^2)$. Consider w in the required form. Let $w = p_1 q_1 \cdots p_k A q_k p_{k+1} q_{k+1} \cdots p_n q_n$, where $n \geq 1$, $1 \leq k \leq n$, $p_j \in (N \cup T)^*$, $q_j \in \{0, 1\}^*$, $1 \leq j \leq n$, and also $p_1 \cdots p_k = x$ and

$p_{k+1} \cdots p_n = y$. Then,

$$w = p_1 q_1 \cdots p_k A q_k p_{k+1} q_{k+1} \cdots p_n q_n$$
$$\Rightarrow_H p_1 q_1 \cdots p_k z q_k p_{k+1} q_{k+1} \cdots p_n q_n = w'$$

in Γ. The required form clearly holds, and thus $S \Rightarrow_H^* w' \Rightarrow_I^t \beta$ in Γ.

The induction step is completed, so Claim 6.3.3.1 holds. ■

Consider $S \Rightarrow^* y$, where $y \in T^*$, in G. By Claim 6.3.3.1, this implies $S \Rightarrow_H^* w \Rightarrow_I^t y$, where $w \in (T \cup \{0,1\})^*$, in Γ. It is obvious that, in such a case, $\Rightarrow_H^*$ behaves exactly the same as $\Rightarrow_H^t$. Thus, $L(G) \subseteq L_t(\Gamma)$. Nonetheless, it is clear that Γ working in the t mode can no longer closely simulate G.

Since the t mode is a restricted case of the $*$ mode, it must hold that $L_t(\Gamma) \subseteq L_*(\Gamma)$. From the proof of Theorem 6.3.1, $L_*(\Gamma) = L(G)$. Therefore, $L_t(\Gamma) \subseteq L(G)$.

As $L(G) \subseteq L_t(\Gamma)$ and $L_t(\Gamma) \subseteq L(G)$, $L_t(\Gamma) = L(G)$. Thus, Theorem 6.3.3 holds. □

Corollary 6.3.4. *The resulting two-component general CD grammar system Γ from the proof of Theorem 6.3.3 changes its components, during every generation of a sentence, no more than once.*

Proof. This proof directly follows the basic idea of Theorem 6.3.3 and Claim 6.3.3.1. Γ always starts the process with the symbol S and the component H, since H is the only component that can generate something from S. If the first derivation does not use any simulated non-context-free rules, then Γ never switches components because the result of such a derivation is already a final sentence. If the result contains verification codes, then Γ switches to the component I that finishes the generation. Since I cannot introduce any new nonterminals of the original grammar, Γ is not able to switch again. □

For the remaining results, we change the second component of the two-component general CD grammar system so it is evenly homogeneous. We show that such a system also works correctly in both the $*$ mode and the t mode.

Theorem 6.3.5. *Let $G = (N, T, P, S)$ be a grammar in Kuroda normal form. Then, there exists a two-component general CD grammar system $\Gamma = (N', T, H, I, S)$ such that H is context-free, $I = \{11 \to 00,\ 0000 \to 2222\}$, and $L_*(\Gamma) = L_t(\Gamma) = L(G)$.*

Proof.
Construction.
Let $G = (N, T, P, S)$. Without any loss of generality, assume that $(N \cup T) \cap \{0,1,2\} = \emptyset$. For $m = 2 + \text{card}(\text{NonContextFree}(P))$, define an injection g from $\text{NonContextFree}(P)$ to $(\{01\}^+\{00\}\{01\}^+ \cap \{01, 00\}^m)$. From G, we construct the two-component general CD grammar system $\Gamma = (N', T, H, I, S)$, where $N' = N \cup \{0,1,2\}$, $I = \{11 \to 00,\ 0000 \to 2222\}$, and H is defined as follows:

(I) For every $AB \to CD \in P$ where $A, B, C, D \in N$,
add $A \to CDg(AB \to CD)$ and $B \to \text{rev}(g(AB \to CD))$ to H.
(II) For every $A \to x \in P$ where $A \in N$ and $x \in (\{\varepsilon\} \cup T \cup N^2)$, add $A \to x$ to H.
(III) Add $2 \to \varepsilon$ to H.

The construction of Γ is completed.

Note that this resembles the construction from Theorem 6.3.1. We only added one new nonterminal and a rule that can erase it. Also the basic idea for the simulation process remains almost the same.

Formal proof (sketch).
First, consider the verification codes. We adjust our terminology and say that the verification code can also contain occurrences of nonterminal 2. Furthermore, the connected verification code now always holds the form $\{01\}\{0,1,2\}^*\{10\}$. Note that only the rule $0000 \to 2222$ can generate 2's and that only the rule $2 \to \varepsilon$ can rewrite these nonterminals further. It follows that the proof of Claim 6.3.1.1 can be trivially adapted for the modified structure, and thus Claim 6.3.1.1 also holds in this system. The following claim introduces a slightly modified version of Claim 6.3.1.2.

Claim 6.3.5.1. *In any reachable sentential form, a verification code can be rewritten only if it can be identified as a connected verification code, sequence of* 0*'s containing a leftover as its substring, or nonterminal* 2*.*

Proof. The proof is analogous to Claim 6.3.1.2. ■

From Claims 6.3.1.1 and 6.3.5.1, it is clear that the purpose of verification codes holds.

Next, consider the $*$ mode. It is obvious that Γ has to switch its components several times if some connected verification code needs to be erased, since the rules of I rewrite only 0's and 1's and the rule from H rewrites 2's. For brevity, let $u \Longrightarrow^l v$ denote the sequence $u \Rightarrow_{k_1} v_1 \Rightarrow_{k_2} \cdots \Rightarrow_{k_l} v_l = v$, $k_j \in \{H, I\}$, $1 \leq j \leq l$. Considering the basic idea in Theorem 6.3.1, we can clearly replace the original derivation sequence $\Rightarrow_I^{2m-1}$ with a new derivation sequence $\Longrightarrow^{6m-1}$. This change can be also straightforwardly applied on Claims 6.3.1.3 and 6.3.1.4 and their proofs. Consequently, $L_*(\Gamma) = L(G)$.

Lastly, consider the t mode. We introduce a modified version of Claim 6.3.3.1. For convenience and brevity, consider the homomorphism $\varphi : (N' \cup T)^* \to (N \cup T)^*$ where $\varphi(a) = a$ and $\varphi(b) = \varepsilon$, for all $a \in (N \cup T)$ and $b \in \{0, 1, 2\}$; and let $u \Longrightarrow^t v$ denote the sequence $u \Rightarrow_{k_1}^t v_1 \Rightarrow_{k_2}^t \cdots \Rightarrow_{k_l}^t v_l = v$, $l \geq 1$, $k_j \in \{H, I\}$, $1 \leq j \leq l$.

Claim 6.3.5.2. *For every $u \in (N \cup T)^*$ and $i \geq 0$, $S \Rightarrow^i u$ in G implies $S \Rightarrow_H^* w \Longrightarrow^t u$ in Γ, where $w \in (N' \cup T)^*$ and $\varphi(w) = u$. Furthermore, we consider w to be generally in the form $w = p_1 q_1 \cdots p_n q_n$, where $n \geq 1$, $p_j \in (N \cup T)^*$, $q_j \in \{0, 1\}^*$, $1 \leq j \leq n$, and every q_j represents a verification code that can be successfully erased on its own.*

Proof. The proof by induction is analogous to Claim 6.3.3.1. ■

Consider $S \Rightarrow^* y$, where $y \in T^*$, in G. By Claim 6.3.5.2, this implies $S \Rightarrow_H^* w \Longrightarrow^t y$, where $w \in (T \cup \{0, 1\})^*$, in Γ. It is again obvious that, in such a case, $\Rightarrow_H^*$ behaves exactly the same as $\Rightarrow_H^t$. Thus, $L(G) \subseteq L_t(\Gamma)$. It is clear that Γ working in the t mode cannot closely simulate G. Furthermore, it is not even possible to bound the number how many times Γ changes its components during the generation of a sentence, since verification codes can be arbitrarily nested and the erasing process needs to constantly switch the components.

As $L_*(\Gamma) = L(G)$, $L(G) \subseteq L_t(\Gamma)$, and $L_t(\Gamma) \subseteq L_*(\Gamma)$, $L_*(\Gamma) = L_t(\Gamma) = L(G)$. Thus, Theorem 6.3.5 holds. □

Corollary 6.3.6. *If the two-component general CD grammar system Γ from the proof of Theorem 6.3.5 works in the $*$ mode, it can closely simulate the original grammar G.*

Proof. The reasoning is the same as for Corollary 6.3.2. For any resulting Γ, we can find a bounded constant k such that for every possible derivation $u \Rightarrow v$ in G there is a k'-step derivation in Γ that gives the same result and $k' \leq k$. Furthermore, for a given Γ, we can easily determine the minimal possible k.

Again, any context-free rule is simulated in one derivation step. The non-context-free rules require two initial derivation steps and the rewriting of the verification code. The length of the rewriting depends on the size of m, and in this case it takes $6m - 1$ steps to complete. The minimal possible k for a given Γ is therefore $6m + 1$. □

6.4 Transformations from General Grammars

This section considers transformations that turn arbitrary general grammars into equivalent two-component general CD grammar systems. Since the previous section already established several transformations from Kuroda normal form, the most straightforward approach would be to convert any general grammar into Kuroda normal form and then use the previous transformations; however, considering the resulting properties of the system, this approach may be undesirable. First, if we want to keep the system close to the original grammar, the transformation into Kuroda normal form already considerably impacts the grammar. Second, the system may generate unnecessarily nested verification codes that can be inconvenient for parallelization. Therefore, we introduce transformations that directly work with general grammars. We say that a transformation from general grammars into two-component general CD grammar systems is *direct* if it keeps the original context-free rules intact and splits the non-context-free rules proportionally to the number of symbols on their left-hand sides.

Theorem 6.4.1. *Let $G = (N, T, P, S)$ be a general grammar such that $\mathrm{alph}(\mathrm{lhs}(p)) \cap T = \emptyset$ for all $p \in P$. Then, there exists its direct transformation into a two-component general CD grammar system $\Gamma = (N', T, H, I, S)$ such that H is context-free, $I = \{11 \to 00,\ 0000 \to \varepsilon\}$, and $L_*(\Gamma) = L_t(\Gamma) = L(G)$.*

Proof.
Construction.
Let $G = (N, T, P, S)$. Without any loss of generality, assume that $(N \cup T) \cap \{0, 1\} = \emptyset$. For $n = \max\{|\,\mathrm{lhs}(p)|\,:\, p \in \mathrm{NonContextFree}(P)\}$ and some $m \geq 3$, define an injection g from $\mathrm{NonContextFree}(P) \times \{1, \ldots, n-1\}$ to $(\{01\}^+\{00\}\{01\}^+ \cap \{01, 00\}^m)$. From G, we construct the two-component general CD grammar system $\Gamma = (N', T, H, I, S)$, where $N' = N \cup \{0, 1\}$, $I = \{11 \to 00,\ 0000 \to \varepsilon\}$, and H is defined as follows:

(I) For every $r\colon X_1 \cdots X_m \to x \in P$ where $m \geq 2$, $X_1, \ldots, X_m \in N$, and $x \in (N \cup T)^*$, add the following rules to H:
$X_1 \to x g(r, 1)$,
$X_2 \to \mathrm{rev}(g(r, 1)) g(r, 2)$,
$\vdots$
$X_{m-1} \to \mathrm{rev}(g(r, m-2)) g(r, m-1)$,
$X_m \to \mathrm{rev}(g(r, m-1))$.

(II) For every $X \to x \in P$ where $X \in N$ and $x \in (N \cup T)^*$, add $X \to x$ to H.

The construction of Γ is completed.

Formal proof (sketch).
To prove the correctness of the above construction, we utilize the previous construction from Theorem 6.3.1, and we go backwards through the transformation of general grammars into Kuroda normal form.

From Theorem 8.3.3.1 in [49], we use the following transformation of a general grammar, $G = (N, T, P, S)$, into an equivalent Kuroda normal form grammar, $G_{\mathrm{KNF}} = (N_{\mathrm{KNF}}, T,$

P_{KNF}, S), which has five distinct steps that modify original rules and add new auxiliary nonterminals. We outline only the necessary basics since details are rather lengthy. All capital letters in the description represent some nonterminals from N_{KNF}. At start, $N_{\mathrm{KNF}} = N$. The five steps follow:

(1) Each occurrence of a terminal, $a \in T$, is replaced with a new nonterminal a', and we add a new rule $a' \to a$.

(2) Every $A_1 \cdots A_m \to B_1 \cdots B_n$, where n and m satisfy $0 \leq n < m$, is replaced with $A_1 \cdots A_m \to B_1 \cdots B_n C_{n+1} \cdots C_m$, where C_{n+1} through C_m denote occurrences of a new nonterminal C. We also add a new rule $C \to \varepsilon$.

(3) Every $A \to B$ is replaced with $A \to BC$ and $C \to \varepsilon$. C is a new nonterminal.

(4) Every $A_1 \cdots A_m \to B_1 \cdots B_n$, where $2 \leq m$ and $3 \leq n$, is repeatedly replaced with $A_1 A_2 \to B_1 C$ and $C A_3 \cdots A_m \to B_2 \cdots B_n$. C is a new nonterminal.

(5) Every $A \to B_1 \cdots B_n$, where $3 \leq n$, is replaced with the standard chain of rules:
$A \to B_1\langle B_2 \cdots B_n\rangle$, $\langle B_2 \cdots B_n\rangle \to B_2\langle B_3 \cdots B_n\rangle$, $\ldots$,
$\langle B_{n-2} \cdots B_n\rangle \to B_{n-2}\langle B_{n-1}B_n\rangle$, $\langle B_{n-1}B_n\rangle \to B_{n-1}B_n$.
$\langle B_2 \cdots B_n\rangle, \ldots, \langle B_{n-1}B_n\rangle$ are new nonterminals.

Let G be a general grammar such that $\mathrm{alph}(\mathrm{lhs}(p)) \cap T = \emptyset$ for all $p \in P$. Let G_{KNF} be a grammar in Kuroda normal form that was created from G according to the above algorithm. And lastly, let Γ_{KNF} be a two-component general CD grammar system that was created from G_{KNF} according to the construction from Theorem 6.3.1. The proofs of Theorems 6.3.1 and 6.3.3 have already shown that we can rewrite nonterminals of G_{KNF} in the sentential form of Γ_{KNF} in any order and that the t mode expands all nonterminals of the original input grammar in one derivation. Therefore, we can, in a backward way, recreate the desired form of rules in Γ from the rules of Γ_{KNF}.

First, consider the original context-free rules of G. They are affected only by the transformation into Kuroda normal form in steps (1), (3), and (5). Therefore, we can easily recreate their original form so that it corresponds with (II). This is possible because each time the transformation into Kuroda normal form splits a rule, it defines some new nonterminal for which only one rule is applicable. Therefore, this rule has to be also eventually applied in Γ_{KNF}, and it causes no issue if both rules are applied together as one in Γ.

Second, consider the context-sensitive rules $A_1 \cdots A_m \to B_1 \cdots B_n$ of G, where $A_1, \ldots, A_m \in N$ and $B_1, \ldots, B_n \in (N \cup T)^*$. First, their right-hand side is affected by (1). Next, they are rewritten with steps (2) and (4) so they have the form: $r_1 \colon A_1 A_2 \to B_1 C_1$, $r_2 \colon C_1 A_3 \to B_2 C_2$, and so on. These remaining context-sensitive rules are then transformed into Γ_{KNF} as: $A_1 \to B_1 C_1 g(r_1)$, $A_2 \to \mathrm{rev}(g(r_1))$, $C_1 \to B_2 C_2 g(r_2)$, $A_3 \to \mathrm{rev}(g(r_2))$, etc. Working backwards, we get the rules of the form: $A_1 \to B_1 \cdots B_n g(r_{m-1}) \cdots g(r_1)$, $A_2 \to \mathrm{rev}(g(r_1))$, $A_3 \to \mathrm{rev}(g(r_2))$, etc. Observe that this creates a nested structure of verification codes. (The same situation inevitably happens in the t mode of Γ_{KNF}.) However, since these are the only rules with the verification codes $g(r_1), \ldots, g(r_{m-1})$, we can safely rearrange the codes in the rules as: $A_1 \to B_1 \cdots B_n g(r_1)$, $A_2 \to \mathrm{rev}(g(r_1))g(r_2)$, $\ldots$, $A_{m-1} \to \mathrm{rev}(g(r_{m-2}))g(r_{m-1})$, $A_m \to \mathrm{rev}(g(r_{m-1}))$. This form then directly corresponds with (I).

Since $L_*(\Gamma_{\mathrm{KNF}}) = L_t(\Gamma_{\mathrm{KNF}}) = L(G)$, it must hold that $L_*(\Gamma) = L_t(\Gamma) = L(G)$. □

The similar result can be easily achieved for the two-component general CD grammar system where $I = \{11 \to 00,\ 0000 \to 2222\}$. Furthermore, it should be also obvious that the other properties from the previous section (close simulation and switching of components) still hold in this more general transformation.

Lastly, we introduce a modification of the above transformation that works with all general grammars. However, it is not possible to directly use our previous approach for grammars that have rules with terminals on their left-hand sides. Consequently, the resulting system may not be able to closely simulate the original general grammar. We say that a transformation is *semi-direct* if it separates terminals from the left-hand side of the rules but otherwise behaves as a direct transformation.

Theorem 6.4.2. *Let $G = (N, T, P, S)$ be a general grammar. Then, there exists its semi-direct transformation into a two-component general CD grammar system $\Gamma = (N', T, H, I, S)$ such that H is context-free, $I = \{11 \to 00,\ 0000 \to \varepsilon\}$, and $L_*(\Gamma) = L_t(\Gamma) = L(G)$.*

Proof. The proof by construction is simple. We use the core idea of step (1) from the transformation of general grammars into Kuroda normal form. First, let $T_{\text{rep}} = \{a : a \in T, a \in \text{alph}(\text{lhs}(r)), r \in P\}$. We replace each occurrence of $a \in T_{\text{rep}}$ in the rules with a new nonterminal a', and we add $a' \to a$ to P. Now, the grammar satisfies the condition for the construction from Theorem 6.4.1. Thus, the construction of Γ is completed. □

Again, the same holds for the system with the evenly homogeneous component.

6.5 Concluding Remarks

We close this chapter by formulating some remarks and open problems. First, take a closer look at the properties of all presented transformations. As already stated in Section 6.1, multi-derivations are performed so that, during a derivation step, the current sentential form may be simultaneously rewritten at several positions, not just at a single position. More formally, let $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$ be a general CD grammar system, n be a positive integer, and $u_i \Rightarrow_{P_k} v_i$, $u_i, v_i \in (N \cup T)^*$, $1 \leq i \leq n$. Then, Γ makes a direct multi-derivation step from $u_1 u_2 \cdots u_n$ to $v_1 v_2 \cdots v_n$, symbolically written as $u_1 u_2 \cdots u_n \ {}_{multi}\Rightarrow_{P_k} v_1 v_2 \cdots v_n$. Based on ${}_{multi}\Rightarrow_{P_k}$, define $L_f(\Gamma)$ by analogy with the definition of $L_f(\Gamma)$ in Section 2.4. Consider the systems constructed in the proofs of Theorems 6.3.1, 6.3.3, 6.3.5, 6.4.1, and 6.4.2. Observe that both of their components H and I always allow the free use of multi-derivations since this cannot disturb the generation process in any way.

Finally, we propose two challenging problems.

(I) Consider the computationally-complete general CD grammar systems presented in this chapter. Can we find a different combination of even more restricted components so that the resulting systems are still computationally complete?

(II) Introduce new restricted transformations of general CD grammar systems so that they characterize some other well-known language families, such as the families of matrix and context-sensitive languages.

Part IV

Conclusion

But apart from the sanitation, the medicine, education, wine, public order, irrigation, roads, the fresh water system, and public health ... what have the Romans ever done for us?

—Monty Python's Life of Brian

Chapter 7

Application Perspectives

In the previous two parts of this thesis, the content was mainly presented in a strictly rigorous way as it appeared in the published papers. In each chapter, we have already mentioned some suggestions for further investigation that are directly linked to the topics in question. Now, we would like to also highlight some broader application perspectives for the obtained results that did not precisely fit into the previous chapters.

7.1 Controlled Discontinuous Reading

First, let us start with the discussion on the reading behavior of jumping finite automata and the types of languages that these models accept. As we have stated in the introduction of this thesis, the main initial motivation behind the jumping concept was the fact that in modern computation methods we often process information in a discontinuous way but classical formal models usually work in a strictly continuous way. Consequently, the description of modern computation methods with classical formal models can be in many cases inadequately complicated. Nonetheless, this does not mean that the proposed jumping finite automata can or should properly cover all needs of discontinuous information processing. Indeed, there are many formal models that try to adequately capture different parts of this phenomenon (see, e.g., the introduction of [57]). This diversity stems from the fact that, with these new models, we are trying to move some complex parts of the behavior of the computation methods into the core structure of the models. As a result, these new models are then more suited for specific tasks rather than for general purpose computing.

Let us take a closer look at the original jumping finite automata (see [57]). Generally, these models can very easily compare quantities of input symbols. Observe the increasing complexity of crossed agreements on the number of symbols in the languages like:

- $\{w \in \{a, b\}^* : |w|_a = |w|_b\}$,
- $\{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$, and
- $\{w \in \{a, b, c, d\}^* : |w|_a = |w|_b = |w|_c = |w|_d\}$.

Since classical formal models need to handle all symbols in the precise order, we quickly get out of the capabilities of even pushdown automata. On the other hand, there is almost no increase in the complexity of the model when we accept these languages with jumping finite automata. Nonetheless, the discontinuous nature of jumping finite automata has its severe drawbacks when we try to accept languages that actually do require some precise reading

order. We can see that JFAs cannot accept even the trivial language $\{ab\}$. This can be partially overcome with GJFAs for finite strings. However, there is no GJFA that could accept the simple regular language $\{a\}^*\{b\}^*$. Moreover, we pay a significant price for this partial reading order with the increased expected parsing complexity. According to [14, 15] and using the standard complexity classes from computational complexity theory, we know that the parsing of JFAs falls into **NL**, whereas the parsing of GJFAs falls into **NP** and there exists a GJFA for which the parsing is NP-complete. In this regard, we can see the behavior of original jumping finite automata mainly as a purely discontinuous reading.

If we take a look at the automata introduced in this thesis, we can say that these parallel versions of jumping finite automata explore controlled discontinuous reading behaviors.

n-Parallel Jumping Finite Automata

Considering models from Chapter 3, we can see that they precisely fit the description of parallelism (P.1) that increases the power of the model. Moreover, the parallel mechanism integrated into the jumping finite automaton model can also partially control the reading order of the input sting. Indeed, we can use the initial splitting of the input string to establish some required fixed order between the separated parts.

Due to Theorems 3.4.1 and 3.4.4, we know that, with any number of reading heads, we can accept all languages accepted by the original jumping finite automata. Now, consider the following languages that cannot be accepted with the original jumping finite automata:

(1) $\{a\}^*\{b\}^*$, $\{a\}^*\{b\}^*\{c\}^*$, $\{a\}^*\{b\}^*\{c\}^*\{d\}^*$,

(2) $\{a^nb^n : n \geq 0\}$, $\{a^nb^nc^n : n \geq 0\}$, and $\{a^nb^nc^nd^n : n \geq 0\}$.

In group (1), the languages are clearly all regular. By contrast, in group (2), the languages get again quickly out of the capabilities of even push automata. It is possible to easily accept all these languages with n-PGJFAs; even if we restrict these automata so that the rules can contain only single symbols. Nonetheless, each distinct section in the input string that occupies a specific position between other sections of the input string requires an additional reading head (see Lemma 3.4.3). Therefore, we can see that this technique of controlling the order of symbols has its limitations. For example, with a finite number of reading heads, we are not able to cover all regular languages.

As we have shown, there is a possibility to use the right n-jumping relation and thus extend the basic behavior of classical finite automata rather than jumping finite automata. In this case, we decrease the capabilities of the discontinuous reading but increase the capabilities of the continuous reading. In this mode, n-PGJFAs are still able to accept all languages introduced in the previous paragraph and also all regular languages, but they are no longer able to accept the previous languages like $\{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$.

It should be a relatively simple task to simulate the parallel behavior of these automata. The model conveniently splits the input into several almost independent parts at the start, and then we need to synchronize only the number of performed steps by the reading heads. The synchronization behavior in this model is also easily reconfigurable, and thus it can be further fine-tuned for the specific needs of the given task. After such a reconfiguration, the changes in the power of the model should be also relatively easy to predict because many different versions of these types of synchronizations were already thoroughly studied in formal grammars.

Double-Jumping Finite Automata

Even though the previous n-PGJFAs are already able to partially control the discontinuous reading, we can see that they can ensure only a relatively simple reading order and that it can potentially require many additional reading heads. Therefore, let us take a look at the advanced reading modes with two reading heads from Chapter 4. We can see that the behavior of the jumping finite automaton model in these modes differs from n-PGJFAs: (1) the input string is not split into separate parts at the start, however, the heads cannot cross each other; (2) each head has its own reading direction; and (3) both heads always follow the same rule in a single step. These changes have a large impact on the jumping behavior and the families of accepted languages, but they help us to better control the discontinuous reading and combine it with the continuous reading. Indeed, we could make these modes more general if we removed (3), but the main focus of our research was to study the core behavior of these modes.

From our initial study, the unrestricted 2-jumping relation does not seem to be particularly interesting for further research. It offers a somewhat similar effect as n-PGJFAs, but the impact of rules on the accepted language is less well-defined because there is no clear point on the input tape which the heads cannot cross. Consequently, we were not able to link this mode to some specific behavior used in modern computation methods.

Considering the right-left and left-right 2-jumping relations, we see that the model loses almost all its jumping capabilities. Indeed, the reading heads can reposition themselves only during the initialization phase. However, this gives us a great control over the reading behavior since the model can still read from two distinct places at once and the position of the heads is always well predictable. Of course, the reduction of jumping capabilities has its drawbacks, and the model can now accept only a subfamily of linear languages. Nonetheless, these 2-jumping relations draw a nice connection to $5' \to 3'$ Watson-Crick finite automata, where this concept can be expanded and studied further.

Lastly, let us take a look at the right-right and left-left 2-jumping relations. On the first glance, their behavior may seem hard to grasp, but under a closer inspection we see here some similarities with methods that ensure data consistency or with other similar types of tasks. In computer science, there are many tasks where several processes may need to work with the same resources together in parallel and where we need to guarantee data consistency. To demonstrate our idea, let us consider database transactions. Typically, in a transaction, we load data from a database, we modify some values, and we save the data back to the database. These three operations together are not atomic, and the database may need to handle other requests in the same time. Thus, when we consider a stream of operations performed by the database, the load and store operations may not follow precisely after each other, but there cannot be a different operation between them that modifies the same data. If we consider the right-right 2-jumping relation, we see that it can accept input strings where the data (symbols) are not in some precise order but where two sequences read by the heads are interlined according to the rule that the first head cannot ever cross the second head. Of course, the model would need to be further appropriately tweaked to precisely match the needs of a real-world task.

Jumping $5' \to 3'$ Watson-Crick Finite Automata

Compared to the previous models, jumping $5' \to 3'$ Watson-Crick finite automata are already constructed with more specific types of tasks in mind. Furthermore, they offer

a significant control of their reading behavior that can be adjusted with the rules in the model. We can explore languages accepted by these automata from two points of view.

First, from a purely theoretical point of view, we can observe how the rules of the model affects the reading behavior. Considering Lemma 5.4.1 and Theorem 5.4.3, we can see that if we use only certain types of rules, we can almost completely disable the jumping behavior of the model. Observe that the meeting point of the heads splits the input string into two parts in which different rules are used. The heads are able to jump in the given part only in cases in which both heads can read some symbols in this part according to the rules of the model. Thus, we can force the heads to read some parts of the input completely continuously. This allows us to accept all regular and even all linear languages. Furthermore, the model is also able to accept some of the more complicated languages with balanced quantities of symbols. Nonetheless, the model is not able to work in a completely discontinuous way, and thus there are more severe restrictions on the form of these languages compare to the original jumping finite automata.

Second, from a more practical point of view, let us consider the biology inspired nature of this model. As we have mentioned in Chapter 2.7, Watson-Crick finite automata work with double-stranded strings, each strand of the string has $5'$ end and $3'$ end, and the strands are combined together from opposite directions. Therefore, if we want both heads to read each strand from $5'$ end to $3'$ end, one head must read from left to right, and the other head must read from right to left. Since this is not a typical behavior of finite automata, it is useful to have special models for these tasks. There are already (full-reading) sensing $5' \to 3'$ Watson-Crick finite automata that work in this way, but their heads can read the input only in a continuous way. Our jumping version of $5' \to 3'$ Watson-Crick finite automata can cover situations where we want to search for sequences in DNA that are not necessarily continuous but rather interlined together from both directions. Of course, one of the imminent follow-up questions is the complexity of a parsing algorithm that would be based on this model. We do not have an answer yet, but we hope that the controlled nature of this discontinuous reading will help us find shortcuts for the parsing process.

7.2 Debt Lemmas

Next, we explore how to use one of the presented proof techniques in a broader context. From the content of Chapters 3, 4, and 5, it is clear that the parallel jumping finite automaton models require different approaches in proofs than classical finite automata that process the input in a continuous way. In Chapters 3 and 4, it was still possible to cleverly adapt the more or less classical proof techniques for the new conditions. However, we can see that the proofs in Chapter 5 sometimes require a significantly different approach. More specifically, let us recall one of the crucial lemmas:

Lemma 5.4.8. *Let L be a language, and let $M = (V, Q, q_0, F, \delta)$ be a jumping $5' \to 3'$ WK automaton. If $L(M) = L$, there exists a constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k.*

For convenience, we name it *the debt lemma*. Traditionally, when we want to show that a language is not accepted by the model in question, we use some form of a pumping lemma (see [77]). Nonetheless, pumping lemmas usually reason about the resulting language families and not about the actual models. This can become limiting in situations where we do not know how the resulting language family precisely looks like, but where we otherwise know a lot about the model that defines it. This situation seems to be quite common for

jumping models because the details of the jumping behavior have a large impact on the resulting language family. If we look at other approaches, it is common in finite automata to say that the model has a finite state control and thus it cannot remember an infinite amount of information. This is, however, only an informal reasoning that is not used in formal proofs. We are not aware of any proof technique that would try to capture such an approach in a formal way. Therefore, we have developed the debt lemma that allows us to take into consideration both the language and also the model.

Even though the debt lemma was designed for the parallel jumping finite automaton model, it is not limited only for these types of models. We believe that it can be easily adapted for any finite automaton model that does not use an infinite storage and that reads the input at least semi-continuously. To demonstrate this in detail we show how to adapt the debt lemma for classical finite automata.

First, we adapt the supporting definitions and lemmas for the new model:

Definition 5.4.5. Let $M = (V, Q, q_0, F, \delta)$ be a jumping $5' \to 3'$ WK automaton. Assuming some states $q, q' \in Q$ and a mutual position of heads $s \in \{\oplus, \ominus\}$, we say that q' is *reachable* from q and s if there exists a configuration (q, s, w_1, w_2, w_3) such that $(q, s, w_1, w_2, w_3) \curvearrowright^* (q', s', w_1', w_2', w_3')$ in M, $s' \in \{\oplus, \ominus\}$, $w_1, w_2, w_3, w_1', w_2', w_3' \in (V \cup \{\#\})^*$.

Definition 7.2.1. Let $M = (V, Q, q_0, F, \delta)$ be a finite automaton. Assuming some states $q, q' \in Q$, we say that q' is *reachable* from q if there exists a configuration qw such that $qw \Rightarrow^* q'$ in M, $w \in V^*$.

Lemma 5.4.6. *For each jumping $5' \to 3'$ WK automaton $M = (V, Q, q_0, F, \delta)$ there exists a constant k such that the following holds. Let $q \in Q$ and $s \in \{\oplus, \ominus\}$ such that $f \in F$ is reachable from q and s. For every computation $\mathcal{C}$ that takes M from $(q_0, \oplus, \varepsilon, w, \varepsilon)$ to (q, s, w_1, w_2, w_3), $w \in V^*$, $w_1, w_2, w_3 \in (V \cup \{\#\})^*$, there exists $w' \in L(M)$ such that M starting with w' can reach q and $s' \in \{\oplus, \ominus\}$ by using the same sequence of $\oplus/\ominus$-reading steps as in $\mathcal{C}$ and the rest of w' can be processed with a limited number of additional steps bounded by k.*

Lemma 7.2.2. *For each finite automaton $M = (V, Q, q_0, F, \delta)$ there exists a constant k such that the following holds. Let $q \in Q$ such that $f \in F$ is reachable from q. For every computation $\mathcal{C}$ that takes M from $q_0 w_1 w_2$ to $q w_2$, $w_1, w_2 \in V^*$, there exists $w' \in L(M)$ such that M starting with w' can reach q by using the same sequence of steps as in $\mathcal{C}$ and the rest of w' can be processed with a limited number of additional steps bounded by k.*

Proof. The proof is trivial. If f is reachable from q, there has to exist some sequence of state transitions from q to f that does not repeat states; this sequence is finite, and its maximum length is bounded by $k' = |Q|$. Assume that the sequence reads $w_3 \in V^*$. Set $w' = w_1 w_3$. Clearly, $q_0 w_1 w_3 \Rightarrow^* q w_3 \Rightarrow^* f$ in M. Thus, $w' \in L(M)$ and there exists $k \leq k'$ for M that bounds the number of additional steps. □

Definition 5.4.7. Let $M = (V, Q, q_0, F, \delta)$ be a jumping $5' \to 3'$ WK automaton, where $V = \{a_1, \ldots, a_n\}$, and let $w \in V^*$. We define the Parikh vector $o = (o_1, \ldots, o_n)$ of processed (read) symbols from w in a configuration $\gamma = (q, s, w_1, w_2, w_3)$ of M reached from an initial configuration $(q_0, \oplus, \varepsilon, w, \varepsilon)$ of M as $o = \Psi_V(w) - \Psi_V(w_1 w_2 w_3)$, $q \in Q$, $s \in \{\oplus, \ominus\}$, $w_1, w_2, w_3 \in (V \cup \{\#\})^*$. Using the Parikh mapping of $L(M)$, we define $\Delta(o) = \{\sum_{i=1}^{n} (m_i - o_i) : (m_1, \ldots, m_n) \in \Psi_V(L(M)),\ m_i \geq o_i,\ 1 \leq i \leq n\} \cup \{\infty\}$. Finally, we define the *debt* of the configuration γ of M as $\min \Delta(o)$.

Definition 7.2.3. Let $M = (V, Q, q_0, F, \delta)$ be a finite automaton, where $V = \{a_1, \ldots, a_n\}$, and let $w \in V^*$. We define the Parikh vector $o = (o_1, \ldots, o_n)$ of processed (read) symbols from w in a configuration $\gamma = qw'$ of M reached from an initial configuration q_0w of M as $o = \Psi_V(w) - \Psi_V(w')$, $q \in Q$, $w' \in V^*$. Using the Parikh mapping of $L(M)$, we define $\Delta(o) = \{\sum_{i=1}^{n}(m_i - o_i) : (m_1, \ldots, m_n) \in \Psi_V(L(M)),\ m_i \geq o_i,\ 1 \leq i \leq n\} \cup \{\infty\}$. Finally, we define the *debt* of the configuration γ of M as $\min \Delta(o)$.

Now, we can adapt the main debt lemma:

Lemma 7.2.4. *Let L be a language, and let $M = (V, Q, q_0, F, \delta)$ be a finite automaton. If $L(M) = L$, there exists a constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k.*

Proof. By contradiction. Assume that there is no constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k. Then, M can accept some $w \in L$ over a configuration for which the debt cannot be bounded by any k. Let $V = \{a_1, \ldots, a_n\}$. Consider any configuration γ of M reached from an initial configuration q_0w of M. Let $o = (o_1, \ldots, o_n)$ be the Parikh vector of processed symbols from w in γ. First, assume that γ contains a state $q \in Q$ from which a final state $f \in F$ is reachable. Then, due to Lemma 7.2.2, there is $w' \in L(M)$ such that $\Psi(w') = (m_1, \ldots, m_n)$, $m_i \geq o_i$, $1 \leq i \leq n$, and $|w'| \leq \sum_{i=1}^{n}(o_i) + k'$, where k' is some constant for M. According to Definition 7.2.3, $w' \in L(M)$ implies $\min \Delta(o) \leq k'$. Second, assume that γ contains a state q from which no final state f is reachable. Then, by Definition 7.2.1, there is no computation that takes M from γ to a final accepting configuration. Thus, when M accepts w, it must be done over configurations with the debt $\leq k'$. However, that is a contradiction with the assumption that M can accept some $w \in L$ over a configuration for which the debt cannot be bounded by any k. □

Finally, we show how the adapted debt lemma can be used in the proof that finite automata cannot define the iconic language $L = \{a^n b^n : n \geq 0\}$.

Theorem 7.2.5. *There is no finite automaton M such that $L(M) = \{a^n b^n : n \geq 0\}$.*

Proof. By contradiction. Let $L = \{a^n b^n : n \geq 0\}$, and let M be a finite automaton such that $L(M) = L$. Due to Lemma 7.2.4, there must exist a constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k. Consider any $k \geq 0$. Let $w = a^{k+1}b^{k+1}$. Clearly, after reading a^k, the debt of the configuration is k, and no further reading is possible. Thus, we can see that M is not able to accept $w = a^{k+1}b^{k+1}$ when the debt of configurations of M is bounded by k. Since, for any k, $w \in L$, there is no constant k for M such that M accepts all $w \in L$ using only configurations that have their debt bounded by k. But that is a contradiction with the assumption that there is a finite automaton M such that $L(M) = \{a^n b^n : n \geq 0\}$. □

Of course, since pumping lemmas for regular languages are well known, it is not necessary to adapt the debt lemma for classical finite automata. However, we can see that the adaptation process is straightforward, and the resulting proof technique is simple to use.

7.3 General Parallel Processing

In Chapter 6, we have introduced several special forms of general CD grammar systems and described their properties which are suitable for parallel rewriting. Nonetheless, we have not yet hinted how the parallel rewriting process could actually look like.

Before we dig deeper, let us take a closer look at parallelism from the hardware perspective. As we have mentioned in the introduction, we want to split some large task into smaller chunks of work in such a way that the chunks can be executed in parallel on separate processing units, and the whole task can thus be computed faster than if it was executed sequentially on a single processing unit. We can also add several details:

- It should be possible to use a variable number of processing units. Indeed, it is common to test the task with a few local processing units and then delegate the real task on a supercomputer.
- It should be possible to split the work equally; otherwise, we are wasting the potential of the used processing units. Consequently, it is desirable for processing units to perform relatively simple instructions.
- The communication between processing units should be kept on minimum; otherwise, a large amount of processing time could be dedicated to the synchronization of data and not to the actual work.

It is easy to construct such a parallel rewriting process for context-free grammars. We can simply wait until there are several nonterminals in the sentential form, then we split the string into several parts so that each part contains at least one nonterminal. Now, all the parts can be safely rewritten in parallel, and the final sentence is just a concatenation of the partial results. It is easy to rewrite nonterminals with context-free rules, there is no need for communication, and the sentential form can be always arbitrarily divided further. Of course, there can be some problematic spots if the grammar generates only a limited number of nonterminals in the sentential form.

The situation gets much more complicated for grammars that use non-context-free rules. First, we cannot safely split the sentential form into several independent parts. Second, we need to search for a whole rewritable sequence and not just for a single nonterminal. Third, even if there are several spots in the sentential form that can be rewritten, it is not clear if we can rewrite several of them in the same time.

Now, consider the new special forms from Chapter 6. We can see several improvements just from the description of their properties. First, these forms have only two non-context-free rules. Moreover, the left-hand sides of rules are short and homogeneous; thus, we only need to look for the sequences of up to four similar symbols. Furthermore, the rewriting of nonterminals 0 and 1 is specifically reserved only for the non-context-free rules, and other nonterminals can be rewritten only with context-free rules. Lastly, any rewritable spot in the sentential form can be rewritten at any time. It remains to be seen if we can meaningfully split the rewriting process in a flexible way and avoid exhaustive communication problems.

Before we describe the remaining properties, let us convert one of the well-known general grammars into the general CD grammar system of the special form.

Consider the following general grammar from [50, Example 11.1]:

$$G = (\{S, A, B, C, D\}, \{a\}, P, S),$$

where P consists of the rules:

$$1\colon\ S \to CAaD, \quad 2\colon\ Aa \to aaA, \quad 3\colon\ AD \to BD, \quad 4\colon\ aB \to Ba,$$
$$5\colon\ CB \to CA, \quad 6\colon\ CA \to A, \quad 7\colon\ AD \to \varepsilon.$$

After the initial phase $S \Rightarrow CAaD$, the grammar G performs sweeps with rules 2, 3, 4, and 5 that always double the number of a's in the sentential form:

$$\begin{aligned} CAaD &\Rightarrow CaaAD \\ &\Rightarrow CaaBD \\ &\Rightarrow CaBaD \\ &\Rightarrow CBaaD \\ &\Rightarrow CAaaD. \end{aligned}$$

The final sweep to the right uses rules 6, 2, and 7 which remove all remaining nonterminals. It is not hard to conclude that $L(G) = \{a^{2^i} : i \geq 1\}$.

We can see that it is impossible to rewrite the sentential form of G in a parallel way since the sweeps require and guarantee that only one specific position is always rewritten; otherwise, the process gets stuck.

Now, according to Theorem 6.4.2, we can construct the two-component general CD grammar system $\Gamma = (N, T, H, I, S)$ from G such that H is context-free, $I = \{11 \to 00,\ 0000 \to \varepsilon\}$, and $L_*(\Gamma) = L_t(\Gamma) = L(G)$. Set $N = \{S, A, B, C, D, a', 0, 1\}$ and $T = \{a\}$, g is an injection from NonContextFree$(P) \times \{1\}$ to $(\{01\}^+\{00\}\{01\}^+ \cap \{01, 00\}^m)$, and H consists of the rules:

$$\begin{aligned} S \to CAa'D, &\quad a' \to a, \\ A \to a'a'Ag(2,1), &\quad a' \to \mathrm{rev}(g(2,1)), \\ A \to BDg(3,1), &\quad D \to \mathrm{rev}(g(3,1)), \\ a' \to Ba'g(4,1), &\quad B \to \mathrm{rev}(g(4,1)), \\ C \to CAg(5,1), &\quad B \to \mathrm{rev}(g(5,1)), \\ C \to Ag(6,1), &\quad A \to \mathrm{rev}(g(6,1)), \\ A \to g(7,1), &\quad D \to \mathrm{rev}(g(7,1)). \end{aligned}$$

Since Γ follows the constructions from the proofs of Theorems 6.4.1 and 6.4.2, it is clear that $L_*(\Gamma) = L_t(\Gamma) = L(G)$.

Finally, we consider the t mode together with the uniform derivations and demonstrate how Γ generates aa. Note that we are also showing the internal direct steps of the t mode.

$$\begin{aligned} S &\ {}_{multi}\Rightarrow_H CAa'D \\ &\ {}_{multi}\Rightarrow_H Ag(6,1)\,\mathrm{rev}(g(6,1))\,\mathrm{rev}(g(2,1))\,\mathrm{rev}(g(7,1)) \\ &\ {}_{multi}\Rightarrow_H a'a'Ag(2,1)g(6,1)\,\mathrm{rev}(g(6,1))\,\mathrm{rev}(g(2,1))\,\mathrm{rev}(g(7,1)) \\ &\ {}_{multi}\Rightarrow_H aag(7,1)g(2,1)g(6,1)\,\mathrm{rev}(g(6,1))\,\mathrm{rev}(g(2,1))\,\mathrm{rev}(g(7,1)) \\ &\ {}_{multi}\Rightarrow_I^+ aag(7,1)g(2,1)\,\mathrm{rev}(g(2,1))\,\mathrm{rev}(g(7,1)) \\ &\ {}_{multi}\Rightarrow_I^+ aag(7,1)\,\mathrm{rev}(g(7,1)) \\ &\ {}_{multi}\Rightarrow_I^+ aa \end{aligned}$$

In the t mode, the rewriting process is clearly divided into two phases.

The first phase works only with component H, and it is essentially classical context-free grammar; thus, it is easy to generate the result in a parallel way. However, we can run into

the same problems as in context-free grammars if the grammar generates only a limited number of nonterminals in the sentential form.

The second phase works only with component I and thus with non-context-free rules. Nonetheless, the nonterminals in the sentential form are quite restricted, and this allows us to optimize the process. As we have mentioned previously, any rewritable spot in the sentential form can be rewritten at any time. Therefore, we can split the sentential form, rewrite everything what is possible, connect the results, and rewrite the remaining spots. Clear indicators for the split can be terminals since they cannot be affected in this stage. However, we can run into problems if all the verification codes are completely nested; that is the situation in our example.

We can also consider the $*$ mode, where the two phases can be mixed. Here we can take advantage of the property that H and I do not rewrite the same nonterminals. The rewriting process can work in a context-free way, and when we see the sequence 11 or 0000 we can simultaneously start the partial processing of the second phase. This approach could actually speed up our previous example.

In conclusion, we can see that these special forms have indeed nice parallel properties. But the usefulness of these properties also depends on the underlying grammar since our special forms closely simulate the original rules. Furthermore, Γ introduces more rules and places where the process can get stuck; therefore, it depends on the fine-tuning of the specific cases whether parallelism can actually speed up the required task.

We can also see here some similarities and potential for novel approaches connected with techniques used in biocomputing that select candidate sequences, connect them, and test them together. In the same fashion, we can split the sentential form in the first phase into several parts, generate final strings with remaining nonterminals 0 and 1, locally test if in these parts the verification codes do not get stuck, grade the parts depending on their required connections to the left and to the right, and then reconnect the fitting results. However, we leave this here only as an interesting side note because a deeper study of this topic is out of the scope of this thesis.

There's no more work. We're destitute. I'm afraid I have no choice but to sell you all for scientific experiments.

—Monty Python's The Meaning of Life

Chapter 8

Summary and Theoretical Perspectives

In this last chapter, we briefly evaluate achieved results and give our final thoughts on the topic. We will not discuss all individual results in detail since each main chapter already has its own concluding remarks, but we will look at the results from a more general perspective.

New Results on Jumping Automata

Considering parallel jumping finite automata, we can see that the newly introduced models match with our goals presented in Chapter 1. All the models fall into the category (PA.2) of parallelism in multi-head finite automata where the heads cooperate to process the single input. In terms of the general categories for parallelism, these models sort of fall into all of them: In category (P.1), parallelism increases the expressive power of the model. We can see that in Chapter 3 the models clearly extend the original jumping finite automaton and have greater expressive powers. In category (P.2), parallelism is a fundamental part of the behavior of the model. In Chapter 5, we work with Watson-Crick models that fall into (P.2) by definition. Lastly, in category (P.3), parallelism splits the work of the task. Since all the models fall into (PA.2), they also naturally fall into (P.3).

With our work, we have pioneered the study of automaton models that combine the parallel and jumping mechanisms. We believe that this area of research nicely supplements the ongoing thorough investigation of the jumping mechanism. Moreover, our results have already inspired some other new models (see [27]). With our results, we have shown that every additional head increases the power of the model and that these automata can be natural counterparts to various kinds of parallel grammars. In our study of 2-jumps, we have shown that, even if we precisely replicate the behavior of right jumps in left jumps, the right-right and left-left 2-jumps define incomparable language families. We have also studied the possibilities of the combined model of jumping and Watson-Crick finite automata, and our results have introduced some new proof techniques like the debt lemma that can be used even outside the scope of jumping finite automata.

In our research, we have always followed the path that looked the most promising to yield new interesting general results. Nonetheless, there are many possibilities how to combine the parallel and jumping mechanism, and thus also many unexplored areas that we were not able to cover. In the previous chapter, we have already hinted some of the areas with potential for future research. For now, jumping finite automata are still primarily interesting from the theoretical point of view. Indeed, they nicely connect different research areas. But

their behavior can be quite wild in more complex cases, their power highly depends on the details of the jumping mechanism, and there are still questions about the complexity of operations with these models. Nonetheless, with enough theoretical knowledge, it can be possible in the future to design and fine tune jumping models so that they properly capture some specific practical problems.

New Results on CD Grammar Systems

Considering the normal forms of grammars and grammar systems, we have introduced several special forms for general CD grammar systems that have interesting parallel properties. The theoretical aspects of these forms are thoroughly described in Chapter 6, and we have also hinted some ideas for practical use in Chapter 7. Note that we are not making the generation process of recursively enumerable languages any simpler, we are just rearranging its parts so that it can be run in a parallel way. Therefore, this approach may not be that interesting for the general case, but it can be useful for more specific cases where we can somehow control and guide the generation process.

Compared to jumping models, this area of research is rather isolated and self-contained. We are not aware of other studies of this type in the basic research in the theory of formal languages. Indeed, classical normal forms are primarily focused only on the very restricted forms of rules, minimum number of non-context-free rules, and minimum number of nonterminals. Therefore, we hope that our effort can spark some interest for this largely unexplored research area.

Bibliography

[1] Beier, S. and Holzer, M. Decidability of Right One-Way Jumping Finite Automata. In: *Developments in Language Theory, DLT 2018*. Springer International Publishing, 2018, p. 109–120. LNCS, vol. 11088.

[2] Beier, S. and Holzer, M. Properties of Right One-Way Jumping Finite Automata. In: *Descriptional Complexity of Formal Systems, DCFS 2018*. Springer International Publishing, 2018, p. 11–23. LNCS, vol. 10952.

[3] Beier, S. and Holzer, M. Properties of Right One-Way Jumping Finite Automata. *Theoretical Computer Science*. 2019, vol. 798, p. 78–94.

[4] Beier, S., Holzer, M. and Kutrib, M. Operational State Complexity and Decidability of Jumping Finite Automata. In: *Developments in Language Theory, DLT 2017*. 2017, p. 96–108. LNCS, vol. 10396.

[5] Beier, S., Holzer, M. and Kutrib, M. Operational State Complexity and Decidability of Jumping Finite Automata. *International Journal of Foundations of Computer Science*. 2019, vol. 30, no. 1, p. 5–27.

[6] Chigahara, H., Fazekas, S. Z. and Yamamura, A. One-way Jumping Finite Automata. In: *The 77th National Convention of IPSJ*. 2015.

[7] Chigahara, H., Fazekas, S. Z. and Yamamura, A. One-way Jumping Finite Automata. *International Journal of Foundations of Computer Science*. 2016, vol. 27, no. 03, p. 391–405.

[8] Csuhaj Varjú, E., Dassow, J., Kelemen, J. and Paun, G. *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach Science Publishers, Inc., 1994.

[9] Csuhaj Varjú, E., Masopust, T. and Vaszil, G. Cooperating Distributed Grammar Systems with Permitting Grammars as Components. *Romanian Journal of Information Science and Technology*. 2009, vol. 12, no. 2, p. 175–189.

[10] Csuhaj Varjú, E., Martín Vide, C. and Mitrana, V. Multiset Automata. In: *Multiset Processing*. Springer Berlin Heidelberg, 2001, p. 69–83. LNCS, vol. 2235.

[11] Eğecioğlu, O., Hegedüs, L. and Nagy, B. Stateless Multicounter $5' \to 3'$ Watson-Crick Automata. In: *Fifth IEEE International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2010*. 2010, p. 1599–1606.

[12] Fazekas, S. Z., Hoshi, K. and Yamamura, A. Enhancement of Automata with Jumping Modes. In: *AUTOMATA 2019: Cellular Automata and Discrete Complex Systems*. 2019, p. 62–76. LNCS, vol. 11525.

[13] Fazekas, S. Z. and Yamamura, A. On Regular Languages Accepted by One-Way Jumping Finite Automata. In: *Eighth Workshop on Non-Classical Models of Automata and Applications, NCMA 2016 (Short Papers)*. 2016, p. 7–14.

[14] Fernau, H., Paramasivan, M. and Schmid, M. L. Jumping Finite Automata: Characterizations and Complexity. In: *Implementation and Application of Automata - 20th International Conference, CIAA 2015*. Springer, 2015, p. 89–101. LNCS, vol. 9223.

[15] Fernau, H., Paramasivan, M., Schmid, M. L. and Vorel, V. Characterization and Complexity Results on Jumping Finite Automata. *Theoretical Computer Science*. 2017, vol. 679, p. 31–52.

[16] Geffert, V. Grammars with Context Dependency Restricted to Synchronization. In: *Mathematical Foundations of Computer Science 1986, MFCS 1986*. 1986, p. 370–378. LNCS, vol. 233.

[17] Geffert, V. Context-Free-Like Forms for the Phrase-Structure Grammars. In: *Mathematical Foundations of Computer Science 1988, MFCS 1988*. 1988, p. 309–317. LNCS, vol. 324.

[18] Geffert, V. A Representation of Recursively Enumerable Languages by Two Homomorphisms and a Quotient. *Theoretical Computer Science*. 1988, vol. 62, no. 3, p. 235–249.

[19] Geffert, V. Normal Forms for Phrase-Structure Grammars. *RAIRO-Theor. Inf. Appl.* 1991, vol. 25, no. 5, p. 473–496.

[20] Goldefus, F., Masopust, T. and Meduna, A. Left-Forbidding Cooperating Distributed Grammar Systems. *Theoretical Computer Science*. 2010, vol. 411, 40–42, p. 3661–3667.

[21] Greibach, S. and Hopcroft, J. Scattered Context Grammars. *Journal of Computer and System Sciences*. 1969, vol. 3, no. 3, p. 233–247.

[22] Grune, D. and Jacobs, C. J. *Parsing Techniques: A Practical Guide*. Secondth ed. Springer, 2008.

[23] Hegedüs, L., Nagy, B. and Eğecioğlu, O. Stateless Multicounter $5' \to 3'$ Watson-Crick Automata: The Deterministic Case. *Natural Computing*. 2012, vol. 11, no. 3, p. 361–368.

[24] Holzer, M., Kutrib, M. and Malcher, A. Multi-Head Finite Automata: Characterizations, Concepts and Open Problems. In: *The Complexity of Simple Programs 2008*. 2009, p. 93–107. EPTCS.

[25] Ibarra, O. H. Simple Matrix Languages. *Information and Control*. 1970, vol. 17, p. 359–394.

[26] Immanuel, S. J. and Thomas, D. G. Two-Dimensional Jumping Finite Automata. *Mathematics for Applications*. 2016, vol. 5, no. 2, p. 105–122.

[27] Immanuel, S. J. and Thomas, D. G. Two-Dimensional Double Jumping Finite Automata. *International Journal of Artificial Intelligence and Soft Computing*. 2017, vol. 6, no. 3, p. 250–264.

[28] Inoue, K., Takanami, I., Nakamura, A. and Ae, T. One-Way Simple Multihead Finite Automata. *Theoretical Computer Science*. 1979, vol. 9, no. 3, p. 311–328.

[29] Křivka, Z., Kučera, J. and Meduna, A. Jumping Pure Grammars. *The Computer Journal*. 2018, vol. 62, no. 1, p. 30–41.

[30] Křivka, Z. and Masopust, T. Cooperating Distributed Grammar Systems with Random Context Grammars as Components. *Acta Cybernetica*. 2011, vol. 20, p. 269–283.

[31] Křivka, Z. and Meduna, A. Jumping Grammars. *International Journal of Foundations of Computer Science*. 2015, vol. 26, no. 6, p. 709–731.

[32] Kocman, R. n-Parallel Jumping Finite Automata. In: *Excel@FIT 2015*. 2015.

[33] Kocman, R., Křivka, Z. and Meduna, A. On Double-Jumping Finite Automata. In: *Eighth Workshop on Non-Classical Models of Automata and Applications, NCMA 2016*. Osterreichische Computer Gesellschaft, 2016, p. 195–210. books@ocg.at, vol. 321.

[34] Kocman, R., Křivka, Z. and Meduna, A. On Double-Jumping Finite Automata and Their Closure Properties. *RAIRO-Theor. Inf. Appl.* 2018, vol. 52, 2-3-4, p. 185–199.

[35] Kocman, R., Křivka, Z. and Meduna, A. General CD Grammar Systems and Their Simplification. *Journal of Automata, Languages and Combinatorics*. 2020, vol. 25, no. 1, p. 37–54.

[36] Kocman, R., Křivka, Z., Meduna, A. and Nagy, B. A Jumping $5' \to 3'$ Watson-Crick Finite Automata Model. *Acta Informatica*. (in review).

[37] Kocman, R. and Meduna, A. On Parallel Versions of Jumping Finite Automata. In: *Proceedings of the 2015 Federated Conference on Software Development and Object Technologies, SDOT 2015*. Springer International Publishing, 2016, p. 142–149. Advances in Intelligent Systems and Computing, vol. 511.

[38] Kocman, R., Nagy, B., Křivka, Z. and Meduna, A. A Jumping $5' \to 3'$ Watson-Crick Finite Automata Model. In: *Tenth Workshop on Non-Classical Models of Automata and Applications, NCMA 2018*. Osterreichische Computer Gesellschaft, 2018, p. 117–132. books@ocg.at, vol. 332.

[39] Kudlek, M., Martín-Vide, C. and Păun, G. Toward a Formal Macroset Theory. In: *Multiset Processing*. Springer Berlin Heidelberg, 2001, p. 123–133. LNCS, vol. 2235.

[40] Kudlek, M. and Mitrana, V. Normal Forms of Grammars, Finite Automata, Abstract Families, and Closure Properties of Multiset Languages. In: *Multiset Processing*. Springer Berlin Heidelberg, 2001, p. 135–146. LNCS, vol. 2235.

[41] Kuperberg, D., Pinault, L. and Pous, D. Cyclic Proofs and Jumping Automata. In: *Foundations of Software Technology and Theoretical Computer Science 2019*. 2019.

[42] Kuske, D. and Weigel, P. The Role of the Complementarity Relation in Watson-Crick Automata and Sticker Systems. In: *Developments in Language Theory, DLT 2004*. 2005, p. 272–283. LNCS, vol. 3340.

[43] Loos, R. and Nagy, B. On the Concepts of Parallelism in Biomolecular Computing. *Triangle 6 (Languages: Bioinspired Approaches)*. 2011, p. 109–118.

[44] Madejski, G. Jumping and Pumping Lemmas and Their Applications. In: *Eighth Workshop on Non-Classical Models of Automata and Applications, NCMA 2016 (Short Papers)*. 2016, p. 25–33.

[45] Madejski, G. Regular and Linear Permutation Languages. *RAIRO-Theor. Inf. Appl.* 2018, vol. 52, 2-3-4, p. 219–234.

[46] Madejski, G. and Szepietowski, A. Membership Problem for Two-Dimensional Jumping Finite Automata. In: *Ninth Workshop on Non-Classical Models of Automata and Applications, NCMA 2017 (Short Papers)*. 2017, p. 33–40.

[47] Mahalingam, K., Raghavan, R. and Mishra, U. K. Watson-Crick Jumping Finite Automata. In: *Theory and Applications of Models of Computation, TAMC 2019*. Springer International Publishing, 2019, p. 467–480. LNCS, vol. 11436.

[48] Masopust, T. and Meduna, A. On Pure Multi-Pushdown Automata that Perform Complete Pushdown Pops. *Acta Cybernetica*. 2009, vol. 19, no. 2, p. 537–552.

[49] Meduna, A. *Automata and Languages: Theory and Applications*. London: Springer, 2000.

[50] Meduna, A. *Formal Languages and Computation: Models and Their Applications*. Auerbach Publications, 2014.

[51] Meduna, A. and Kolář, D. Homogenous Grammars with a Reduced Number of Non-Context-Free Productions. *Information Processing Letters*. 2002, vol. 81, no. 5, p. 253–257.

[52] Meduna, A. and Masopust, T. Self-Regulating Finite Automata. *Acta Cybernetica*. 2007, vol. 18, no. 1, p. 135–153.

[53] Meduna, A. and Soukup, O. Jumping Scattered Context Grammars. *Fundamenta Informaticae*. 2017, vol. 152, no. 1, p. 51–86.

[54] Meduna, A. and Soukup, O. *Modern Language Models and Computation: Theory with Applications*. Springer, 2017.

[55] Meduna, A. and Švec, M. *Grammars with Context Conditions and Their Applications*. Wiley, 2005.

[56] Meduna, A., Švec, M. and Kopeček, T. Equivalent Language Models that Closely Simulate One Another and Their Illustration in Terms of L Systems. *International Journal of Computer Mathematics*. 2007, vol. 84, no. 11, p. 1555–1566.

[57] Meduna, A. and Zemek, P. Jumping Finite Automata. *International Journal of Foundations of Computer Science*. 2012, vol. 23, no. 7, p. 1555–1578.

[58] Meduna, A. and Zemek, P. *Regulated Grammars and Automata*. Springer, 2014.

[59] Nagy, B., Hegedüs, L. and Eğecioğlu, O. Hierarchy Results on Stateless Multicounter $5' \to 3'$ Watson-Crick Automata. In: *Advances in Computational Intelligence: 11th International Work-Conference on Artificial Neural Networks, IWANN 2011*. Springer, 2011, p. 465–472. LNCS, vol. 6691.

[60] Nagy, B. On $5' \to 3'$ Sensing Watson-Crick Finite Automata. In: *The 13th International Meeting on DNA Computing (DNA13)*. 2007, p. 327–336.

[61] Nagy, B. On $5' \to 3'$ Sensing Watson-Crick Finite Automata. In: *DNA Computing: 13th International Meeting on DNA Computing, DNA13*. Springer, 2008, p. 256–262. LNCS, vol. 4848.

[62] Nagy, B. On a Hierarchy of $5' \to 3'$ Sensing WK Finite Automata Languages. In: *Computability in Europe 2009: Mathematical Theory and Computational Practice, CiE 2009*. 2009, p. 266–275.

[63] Nagy, B. $5' \to 3'$ Sensing Watson-Crick Finite Automata. In: Fung, G., ed. *Sequence and Genome Analysis II – Methods and Applications*. IConcept Press, 2010, p. 39–56.

[64] Nagy, B. A Class of 2-Head Finite Automata for Linear Languages. *Triangle*. 2012, 8 (Languages: Mathematical Approaches), p. 89–99.

[65] Nagy, B. On a Hierarchy of $5' \to 3'$ Sensing Watson-Crick Finite Automata Languages. *Journal of Logic and Computation*. 2013, vol. 23, no. 4, p. 855–872.

[66] Nagy, B. and Kovács, Z. On Simple $5' \to 3'$ Sensing Watson-Crick Finite-State Transducers. In: *Eleventh Workshop on Non-Classical Models of Automata and Applications, NCMA 2019*. 2019, p. 155–170.

[67] Nagy, B. and Otto, F. Two-Head Finite-State Acceptors with Translucent Letters. In: *SOFSEM 2019: Theory and Practice of Computer Science*. Springer International Publishing, 2019, p. 406–418. LNCS, vol. 11376.

[68] Nagy, B. and Otto, F. Linear Automata with Translucent Letters and Linear Context-Free Trace Languages. *RAIRO-Theor. Inf. Appl.* 2020, vol. 54.

[69] Nagy, B. and Parchami, S. On Deterministic Sensing $5' \to 3'$ Watson-Crick Finite Automata: A Full Hierarchy in 2detLIN. *Acta Informatica*. 2020.

[70] Nagy, B., Parchami, S. and Mir-Mohammad-Sadeghi, H. A New Sensing $5' \to 3'$ Watson-Crick Automata Concept. In: *Proceedings 15th International Conference on Automata and Formal Languages, AFL 2017*. Open Publishing Association, 2017, p. 195–204. EPTCS, vol. 252.

[71] Parchami, S. and Nagy, B. Deterministic Sensing $5' \to 3'$ Watson-Crick Automata Without Sensing Parameter. In: *Unconventional Computation and Natural Computation, UCNC 2018*. 2018, p. 173–187. LNCS, vol. 10876.

[72] Păun, G., Rozenberg, G. and Salomaa, A. *DNA Computing: New Computing Paradigms*. Springer-Verlag Berlin Heidelberg, 1998.

[73] Rosebrugh, R. D. and Wood, D. A Characterization Theorem for n-Parallel Right Linear Languages. *Journal of Computer and System Sciences*. 1973, vol. 7, p. 579–582.

[74] Rosebrugh, R. D. and Wood, D. Image Theorems for Simple Matrix Languages and n-Parallel Languages. *Mathematical Systems Theory*. 1974, vol. 8, no. 2.

[75] Rosebrugh, R. D. and Wood, D. Restricted Parallelism and Right Linear Grammars. *Utilitas Mathematica*. 1975, vol. 7, p. 151–186.

[76] Rosenberg, A. L. On Multi-Head Finite Automata. In: *6th Annual Symposium on Switching Circuit Theory and Logical Design, SWCT 1965*. 1965, p. 221–228.

[77] Rozenberg, G. and Salomaa, A. *Handbook of Formal Languages, Vol. 1: Word, Language, Grammar*. Springer-Verlag, 1997.

[78] Rozenberg, G. and Salomaa, A. *Handbook of Formal Languages, Vol. 2: Linear Modeling: Background and Application*. Springer-Verlag, 1997.

[79] Savitch, W. J. How to Make Arbitrary Grammars Look Like Context-Free Grammars. *SIAM Journal on Computing*. 1973, vol. 2, no. 3, p. 174–182.

[80] Siromoney, R. On Equal Matrix Languages. *Information and Control*. 1969, vol. 14, no. 2, p. 135–151.

[81] Siromoney, R. Finite-Turn Checking Automata. *Journal of Computer and System Sciences*. 1971, vol. 5, no. 6, p. 549–559.

[82] Syropoulos, A. Mathematics of Multisets. In: *Multiset Processing*. Springer Berlin Heidelberg, 2001, p. 347–358. LNCS, vol. 2235.

[83] Ďuriš, P. and Hromkovič, J. One-Way Simple Multihead Finite Automata are not Closed Under Concatenation. *Theoretical Computer Science*. 1983, vol. 27, no. 1, p. 121–125.

[84] Vorel, V. Two Results on Discontinuous Input Processing. In: *Descriptional Complexity of Formal Systems: 18th IFIP WG 1.2 International Conference, DCFS 2016*. Springer International Publishing, 2016, p. 205–216. LNCS, vol. 9777.

[85] Vorel, V. Two Results on Discontinuous Input Processing. *Journal of Automata, Languages and Combinatorics*. 2017, vol. 22, 1–3, p. 189–203.

[86] Vorel, V. On Basic Properties of Jumping Finite Automata. *International Journal of Foundations of Computer Science*. 2018, vol. 29, no. 1, p. 1–15.

[87] Wang, Q. and Li, Y. Jumping Restarting Automata. In: *Tenth Workshop on Non-Classical Models of Automata and Applications, NCMA 2018*. Osterreichische Computer Gesellschaft, 2018, p. 181–196. books@ocg.at, vol. 332.

[88] Wood, D. Properties of n-Parallel Finite State Languages. *Utilitas Mathematica*. 1973, vol. 4, p. 103–113.

[89] Wood, D. m-Parallel n-Right Linear Simple Matrix Languages. *Utilitas Mathematica*. 1975, vol. 8, p. 3–28.

[90] Wood, D. n-Linear Simple Matrix Languages and n-Parallel Linear Languages. *Rev. Roum. de Math. Pures et Appl.* 1977, p. 408–412.

[91] Wood, D. *Theory of Computation: A Primer*. Boston: Addison-Wesley, 1987.