BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

PHD THESIS

Brno, 2021

Ing. Jiří Kučera

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

NEW PARALLEL AND REGULATED AUTOMATA AND GRAMMARS

NOVÉ PARALELNÍ A REGULOVANÉ AUTOMATY A GRAMATIKY

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR AUTOR PRÁCE	**Ing. JIŘÍ KUČERA**
SUPERVISOR ŠKOLITEL	**prof. RNDr. ALEXANDER MEDUNA, CSc.**

BRNO 2021

Abstract

This thesis introduces and studies four new language models with focus on regulation and parallelism in automata and grammars. First, *state-synchronized automata systems*, present systems consisting of a finite number of pushdown automata controlled by words of a control language over a set of states. Second, *unlimited deep pushdown automata*, are a modification of deep pushdown automata with no restrictions imposed on the depth of expansion on the pushdown. Third, *jumping pure grammars*, introduces jumping grammars with no nonterminal symbols. The last one, *k#$-rewriting systems*, extends #-rewriting systems with additional pushdown memory.

The contents of this thesis are divided into three parts. The first part outlines the motivation for introduction of the studied language models and puts them into the context of related formal language theory areas. Furthermore, it gives an overview of the structure of the thesis, reviews fundamental notions of formal language theory and gives a survey of current knowledge related to the subject of research. The second part presents the core of this thesis. Here, formal definitions of all newly introduced language models are given and their expressive power is studied. Finally, this thesis is concluded with a summary of achieved theoretical results as well as related open problem areas, and an outline of the possibilities for further research along with a sketch of possible applications.

Keywords

formal language theory, 0L languages, pure context-free languages, recursively enumerable languages, finite index, infinite hierarchy, automata, pushdown, pushdown automata, automata systems, deep pushdown automata, state-synchronized automata systems, unlimited deep pushdown automata, grammars, pure grammars, jumping grammars, jumping pure grammars, rewriting, parallel rewriting, jumping rewriting, rewriting systems, #-rewriting systems, *k*#$-rewriting systems

Bibliographic Citation

KUČERA, Jiří. *New Parallel and Regulated Automata and Grammars*. Brno, 2021. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Meduna Alexander.

Abstrakt

Tato práce zkoumá a zavádí čtyři nové jazykové modely se zaměřením na regulované a paralelní verze automatů a gramatik. První z modelů, *stavově synchronizované systémy automatů*, zavádí systémy složené z konečného počtu zásobníkových automatů jejichž činnost je řízena slovy z řídícího jazyka nad množinou stavů. Druhý model, *neomezené hluboké zásobníkové automaty*, je variantou hlubokých zásobníkových automatů z nichž bylo sejmuto omezení kladené na hloubku expanze na zásobníku. Třetí model, *skákající čisté gramatiky*, zavádí skákající gramatiky bez neterminálních symbolů. Poslední model, *k#$-přepisující systémy*, rozšiřuje #-přepisující systémy o přídavnou zásobníkovou paměť.

Text této práce je členěn do tří částí. První část uvádí motivaci k zavedení studovaných jazykových modelů a zasazuje je do kontextu souvisejících oblastí teorie formálních jazyků. Je zde také uveden přehled o celkové organizaci práce, základních pojmů teorie formálních jazyků a současných poznatků souvisejících s předmětem výzkumu. Druhá část tvoří jádro této práce. Jsou v ní formálně definovány všechny nově zavedené jazykové modely a studována jejich vyjadřovací síla. Poslední část uzavírá tuto práci souhrnem získaných teoretických výsledků společně se souvisejícími otevřenými problémy a nastiňuje cesty dalšího výzkumu spolu s výhledy na možná využití.

Klíčová slova

teorie formálních jazyků, 0L jazyky, čisté bezkontextové jazyky, rekurzivně vyčíslitelné jazyky, konečný index, nekonečná hierarchie, automaty, zásobník, zásobníkové automaty, systémy automatů, hluboké zásobníkové automaty, stavově-synchronizované systémy automatů, neomezené hluboké zásobníkové automaty, gramatiky, čisté gramatiky, skákající gramatiky, skákající čisté gramatiky, přepisování, paralelní přepisování, skákající přepisování, přepisující systémy, #-přepisující systémy, *k*#$-přepisující systémy

Bibliografická Citace

KUČERA, Jiří. *Nové Paralelní a Regulované Automaty a Gramatiky*. Brno, 2021. Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Školitel Meduna Alexander.

Rozšířený Abstrakt

Mezi nejčastější způsoby, kterými lze v teorii formálních jazyků formálně popsat jazyk, patří gramatiky, automaty, a algebraický zápis. Gramatiky pracují na principu generování slov daného jazyka z počátečního symbolu gramatiky postupnou aplikací daných pravidel. Automaty, naproti tomu, pracují opačným způsobem. Automaty čtou vstupní slovo symbol po symbolu na základě čehož mění svoji vnitřní konfiguraci. Slovo patří do jazyka přijímaného automatem tehdy a jen tehdy došlo-li k přečtení všech symbolů slova na vstupu a automat se nachází v koncovém stavu. Automaty a gramatiky jsou někdy souhrně označovány jako *přepisující systémy*, neboť při jejich činnosti dochází k přepisování jejich vnitřní konfigurace.

Dle Chomského klasifikace[9] lze rozdělit gramatiky do dvou skupin na gramatiky s bezkontextovými pravidly a gramatiky s kontextovými pravidly. Výhodou gramatik s bezkontextovými pravidly je jejich snadný a intuitivní způsob užití při definici jazyka, nevýhodou pak slabší vyjadřovací síla. Gramatiky s kontextovými pravidly tímto nedostatkem netrpí, ovšem za cenu jejich komplexnosti použití[88]. Vzhledem ke skutečnosti, že svět není bezkontextový, teorie formálních jazyků se začala zabývat způsoby jak zvýšit vyjadřovací sílu bezkontextových gramatik. Jedním z cílů, jak toho bylo dosaženo, bylo zavedení tzv. *regulovaného přepisování*[21], kdy je aplikace pravidla podmíněna splněním dotatečných požadavků, např. výskytem symbolů ve větné formě nebo předchozí aplikací jiného pravidla. Vyjadřovací sílu automatů a gramatik lze také ovlivnit jejich seskupením[4, 7, 15, 18] do systémů. V takovém systému představují automaty a gramatiky tzv. komponenty, které spolu mohou navíc vzájemně komunikovat podle předem daného protokolu.

Tato práce obohacuje teorii formálních jazyků o čtyři nové jazykové modely. Prvním modelem jsou stavově synchronizované automatové systémy, publikované v [53]. Stavově synchronizovaný automatový systém je definován jako systém složený obecně z n zásobníkových automatů jako komponent a řídícího jazyka, jehož slova jsou utvořena ze stavů jednotlivých komponent. Výpočetní krok lze v tomto systému provést tehdy a jen tehdy, mohou-li všechny komponenty provést svůj výpočetní krok současně a zároveň nachází-li se slovo utvořené z aktuálních stavů komponent v řídícím jazyce. Ve stavově synchronizovaném automatovém systému je vstupní slovo čteno pouze první komponentou.

Druhým modelem jsou neomezené hluboké zásobníkové automaty. Jejich činnost je založena na principu hlubokých zásobníkových automatů[68], ovšem s tím rozdílem, že hloubka expanze symbolu na zásobníku není nijak omezena. V principu je tak možné expandovat libovolný symbol na zásobníku. Neomezené hluboké zásobníkové automaty jsou prezentovány ve dvou variantách—absolutně neomezené hluboké zásobníkové automaty a relativně neomezené hluboké zásobníkové automaty. U první varianty, publikované v [54], dochází k expanzi prvního vhodného symbolu nejblíže vrcholu zásobníku. U druhé varianty, dosud nepublikované, dochází k expanzi symbolu relativně od místa předchozí expanze.

Třetí model, skákající čisté gramatiky, publikované v [49], jsou tvořeny čistými gramatikami nad nimiž jsou definovány čtyři derivační módy—klasický sekvenční, klasický paralelní, skákající sekvenční a skákající paralelní. Skokem se v kontextu skákajících

gramatik rozumí vymazání levé části aplikovaného pravidla z větné formy a následným zapsáním části pravé na libovolné místo v téže větné formě v rámci jednoho derivačního kroku. Pojem paralelní potom označuje, že musejí být v rámci jednoho derivačního kroku přepsány veškeré symboly větné formy současně.

Posledním modelem, publikovaným v [48], jsou k#\$-přepisující systémy. Jedná se o #-přepisující systémy[52] rozšířené o přídavnou zásobníkovou paměť oddělenou od konfigurace #-přepisujícího systému symbolem \$. Nachází-li se nalevo od symbolu \$ nedostatek nebo nadbytek symbolů # takový, že nelze aplikovat žádné přepisovací pravidlo, dojde za pomocí speciálních pravidel buď k přesunu dostatečného počtu neterminálů a jejich transformaci na symboly # nebo k přesunu nadbytečného počtu symbolů # a jejich transformaci na neterminály z/do zásobníkové paměti napravo od symbolu \$.

U všech modelů byla v práci studována jejich vyjadřovací síla. Získané výsledky jsou podloženy matematickými důkazy. Stavově synchronizované automatové systémy a neomezené hluboké zásobníkové automaty jsou Turingovsky úplné. Skákající čisté gramatiky byly studovány s bezkontextovým tvarem pravidel. Použití rozdílných derivačních módů u skákajících čistých gramatik vede k množství rozdílných rodin jazyků, jejichž vztah je znázorněn na Obrázku 1. k#\$-přepisující systémy mají stejnou vyjadřovací sílu jako k-omezené stavové gramatiky. Mimo jiné bylo dokázáno, že rodina jazyků generovaných programovanými gramatikami indexu k je vlastní podmnožinou rodiny jazyků generovaných k-omezenými stavovými gramatikami.

Declaration

I hereby declare that this thesis is my authorial work that has been created under the supervision of prof. RNDr. Alexander Meduna, CSc. It is based on papers [48, 49, 53, 54] which I have written in cooperation with my supervisor. Furthermore, paper [54] and papers [48, 49] were reviewed by my colleagues Ing. Ondřej Soukup, PhD. and Ing. Zbyněk Křivka, PhD., respectively. Chapter 5 and Chapter 7 also contain extra content that was not included in papers [48, 54] due to the size limit. More precisely, Chapter 5 presents relatively unlimited deep pushdown automata and Chapter 7 contains the full versions of proofs from [48]. All sources of information used in this thesis were properly cited.

Jiří Kučera
April 26, 2021

Acknowledgments

This thesis was supported by the following grants: European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070); TAČR grant TE01020415; BUT grant FIT-S-14-2299; 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; BUT grant FIT-S-17-3964; and BUT grant FIT-S-20-6293.

I would like to thank prof. RNDr. Alexander Meduna, CSc. for his supervision of this work, for his inspiring and valuable consultations, and for his friendly leading. I would also like to thank my colleagues from the university, namely Ing. Zbyněk Křivka, PhD., Ing. Ondřej Soukup, PhD., Ing. Jakub Martiško, Ing. Radim Kocman, and Ing. Dominika Klobučníková, for their inspiring discussion about formal languages. Last, but not least, I would like to thank my family for their love, support, and patience.

Contents

Part I
Introduction and State of the Art

This part gives an introduction to this thesis. It puts the topic of the research into the context of formal language theory, reviews the mathematical background to avoid possible confusion, and gives a survey of related rewriting systems. The contents are organized into three chapters.

Chapter 1, the introductory chapter of this thesis, briefly introduces formal language theory with focus on regulated, cooperative, and parallel rewriting. It also explains the motivation for introduction and examination of new language models presented in the following part and outlines how the contents are organized.

Chapter 2 reviews notation and terminology used in the following chapters to make this thesis self-contained. Specifically, it gives an overview of fundamental notions of formal language theory with related mathematical background and reviews the language models concerning the Chomsky hierarchy of language families.

Chapter 3 gives a survey of the current state of knowledge. It discusses rewriting systems related to the topic of research, especially those utilizing regulated, cooperative, and parallel rewriting. Emphasis is laid on how rewriting works in these systems, as well as what its impact on their descriptive/expressive power is.

Chapter 1
Introduction

During the last decades, information technology has become an essential part of almost every aspect of everyday life in modern human civilization. With the ability of information exchange and processing, information technology plays a crucial role in research, industry, health care, and many others areas. However, none of this would be possible without intensive research in areas concerning information technology itself, mainly in mathematics, physics, chemistry, and computer science. One of the areas that have a big impact on information technology development is the *formal language theory*.

The formal language theory (see [86–88]) studies ways of expressing languages mathematically along with their properties. It provides valuable tools that can be used everywhere where a problem can be represented by a formal language, such as constructing a compiler (see [3]), defining a protocol for information exchange, but also in biology and linguistic. The most common methods used to define languages formally are grammars, automata, and algebraic notation. In principle, grammars work as generative devices. A grammar generates a language from its initial symbol by consecutive applications of rules. In contrast, automata are devices for language recognition. An automaton reads an input word symbol by symbol, changing its state accordingly. If the automaton enters its final state and all input symbols have been read, the input word belongs to the language recognized by the automaton. Grammars and automata are sometimes commonly referred to as *rewriting systems* because when generating or recognizing a language, grammars and automata[1] *rewrite* their inner state[2].

Work on formal language theory started evolving when Noam Chomsky published his two famous papers *Three models for the description of language* (see [9]) and *On certain formal properties of grammars* (see [10]). In his work, Chomsky restricts the form of rules of phrase-structured grammars, which led to the hierarchy of language families recently known as the Chomsky hierarchy. The area of formal language theory that focuses only on languages from the Chomsky hierarchy is referred as the *classical* formal language theory.

According to forms of their rules, grammars from the Chomsky hierarchy can be divided into two groups: grammars with context-free rules and grammars with non-context-free rules. Both types of grammars have both advantages and disadvantages. When using a

[1] There are two ways of defining automata. The first one (see [31]) defines automata using a transition function. The second one (see [66, 89]) defines automata similarly to grammars: instead of a transition function, it uses a set of rules where both sides of a rule are sequences of symbols.

[2] A sentential form for grammars and a configuration for automata.

non-context-free rule, a sequence of symbols is rewritten; in the case of a context-free rule it is just one symbol. This difference becomes evident when defining a language. Non-context-free rules are not trivial to work with. Their left-hand sides must be designed very carefully in order to ensure that the resulting grammar really generates the desired language.

Context-free rules, on the other hand, can be used in a much more intuitive way but their expressive power is relatively weak. Considering these facts, it is then no surprise that finding new way of using grammars with context-free rules in ways that can generate non-context-free languages has become a vivid research topic of the *modern* formal language theory. The most known achievement in this area is *regulated rewriting*.

Regulated Rewriting

Regulated rewriting (see [21, 72]) studies rewriting systems extended with additional mechanism that control the way their rules are applied with the aim of improving their expressive power. To demonstrate the principle on context-free grammars, when performing a derivation step, a rule can be applied only if its left-hand side matches a symbol in the current sentential form; in case of more than one match, any of the matching symbols can be rewritten. However, when context-free grammars are regulated, additional criteria must be fulfilled. Such a criterion can be for instance a prescribed order of application of rules. This can be achieved, for example, by organizing rules into sequences called *matrices* (see [1]), by specifying successors to every rule (see [82], the original paper on *programmed grammars*), or by specifying a pattern in which rules should be applied (see [5]). Another criterion, used in *random context grammars* (see [95]), is the presence or absence of specific symbols in sentential form.

Equipped with a mechanism that controls the application of rules, context-free grammars are able to generate languages that are not context-free. Moreover, some regulated grammars are capable of generating any recursively enumerable language (see [21, 72] for examples).

Besides regulated grammars, there also exist regulated automata, but research in this area is not as intensive as in the case of regulated grammars. Some representatives of regulated automata are *regulated pushdown automata* (see [43, 44]), *self-regulating automata* (see [61, 62]), or *jumping finite automata* (see [71]).

Parallel and Cooperative Rewriting

Till now, only rewriting systems that work sequentially have been mentioned. In the real world, however, most processes are done simultaneously. This can be seen in the growth of red algae, which was the motivation behind *L-systems* (see [56, 57]). L-systems, which were introduced in [56, 57], later started being referred to as 0L systems due to intensive research in this area (see Chapter *L Systems* in [88]). In 0L systems, context-free rules are applied to simultaneously rewrite every symbol in a sentential form during one rewriting step. As a consequence, 0L systems are capable of expressing languages that are not context-free; however, there exist context-free languages that cannot be expressed by 0L systems.

Another trait of real-world processes is that besides being performed simultaneously, they also communicate with their environment. Communication of entities that work together to reach their common goal became the inspiration behind the introduction of *grammar and automata systems*. *Cooperating/distributed grammar systems* (see [14]), for example, were inspired by problem-solving method called the blackboard model. In these systems, grammars are treated as agents that work on a shared sentential form (a *blackboard*). The cooperation is reached via derivation modes which determine how long one agent can work continuously on the sentential form.

Besides cooperative/distributive grammar systems, where only one grammar can actively work on a sentential form during one derivation step, *parallel communicating grammar systems* (see [91]) are worth mentioning. As opposed to cooperating/distributed grammar systems, every grammar—called a *component*—works on its own sentential form. Communication in these grammar systems is handled via *query symbols* – at some point during a derivation, one of the components produces a special symbol that is subsequently replaced by the sentential form of different component.

With their connection to many research areas, such as DNA computing, parallel computing or distributed computing, grammar and automata systems have become a vivid research topic in the modern formal language theory (see [15]). *Multi-generative grammar systems* (see [58]) are another significant result in this area. Grammars in these systems can be synchronized by either rules[3] or nonterminal symbols. An interesting property of multi-generative grammar systems is that they generate *n-languages*—sets containing n-tuples of words—which makes them suitable for practical tasks, e.g. compiling source program to multiple targets in one step (see Chapter 7 in [58]).

Principles similar to the ones used in grammar systems have also been used in automata systems. To list several examples, automata in *distributed pushdown automata systems* (see [18]) use protocols for cooperation similar to grammars in cooperating/distributed grammar systems; *parallel communicating pushdown automata systems* (see [17]) and *parallel communicating finite automata systems* (see [60]) use query pushdown symbols and query states, respectively, and *n-accepting restricted automata systems* (see [6]) were introduced as a counterpart to multi-generative grammar systems.

This list of grammar and automata systems is not comprehensive. There exist other types of grammar and automata systems, such as grammar and automata systems with *teams* (see [4, 34]).

Motivation

The aim of research behind this thesis was to establish new language models with a focus on parallel and regulated rewriting and investigate their expressive power with respect to the

[3] A similar approach can be noticed also in *scattered context grammars* (see [28]), where rules are ordered into tuples and it is then possible to transmit information between separated parts of the sentential form.

Chomsky hierarchy of languages. More precisely, this thesis presents four new language models:

I. *State-synchronized automata systems*, published in [53], are systems consisting generally of n pushdown automata. The communication between automata is done via their states. The idea of using states in the communication process between automata in automata systems has been examined before (see e.g. [6, 60]); however, in state-synchronized automata systems states are used in a more elegant and natural way. In state-synchronized automata systems, the synchronization is handled by finite control language containing words formed from states of specific automata. A computation step can be performed if and only if all automata can make their moves simultaneously and their states form a word of control language.

II. *Unlimited deep pushdown automata* are a modification of deep pushdown automata. Deep pushdown automata (see [68]) allow expansion of pushdown symbols deeper on their pushdown, but the depth of expansion is limited. As a result, they are able to accept languages that are not context-free, but not every context-sensitive language. It is natural to modify deep pushdown automata by not imposing a limit on the depth of expansion. Unlimited deep pushdown automata are presented in two variants: *absolutely unlimited deep pushdown automata*, published in [54], and *relatively unlimited deep pushdown automata* (not yet published). The main difference between them is the way the pushdown symbol that should be expanded is chosen. Absolutely unlimited deep pushdown automata expand the first suitable pushdown symbol at any depth. Relatively unlimited deep pushdown automata keep the position on the pushdown as a part of their configuration. A pushdown symbol is expanded relatively to this position and the position is updated.

III. *Jumping pure grammars*, published in [49], work like jumping grammars (see [51]) while being based on *pure grammars* (see [26, 63, 89]). As jumping versions of automata and grammars became a vivid topic in the modern formal language theory (see [8, 22, 23, 40, 42, 51, 69, 71]), it is natural to contribute to this topic by introducing the jumping versions of pure grammars. Jumping pure grammars were studied with four modes of derivations: classical sequential, classical jumping, parallel sequential, and parallel jumping.

IV. k#\$-*rewriting systems*, published in [48], extend #-rewriting systems (see [52]) with additional pushdown-like storage. As shown in Section 4 in [52], the language family generated by #-rewriting systems of index k coincides with the language family generated by programmed grammars of the same index. The motivation behind the idea of introducing k#\$-rewriting systems was twofold: (a) to prove the conjecture that #-rewriting systems of index k extended with additional pushdown-like storage generate the language family which coincides with the language family generated by k-limited state grammars and (b) to show that if the conjecture from (a) holds the language family generated by programmed grammars of index k is a proper subset of the language family generated by k-limited state grammars.

Organization

After the introduction given in this chapter, Chapter 2 reviews fundamental mathematical terms and notation used in this thesis to eliminate the risk of confusion. Chapter 3 discusses rewriting systems studied so far related to the subject of research carried out within this thesis.

Starting with Chapter 4, the following four chapters present the core of this thesis. Chapter 4 defines *state-synchronized automata systems* and studies their language properties with respect to the type of their components. Chapter 5 defines and studies *absolutely* and *relatively unlimited deep pushdown automata*. In Chapter 6, *jumping pure grammars* are defined and examined. Finally, *k#$-rewriting systems* are defined and examined in Chapter 7. All four chapters also demonstrate defined formal models by showing examples and give formal proof of all stated theorems.

Finally, Chapter 8 concludes this thesis by summarizing achieved theoretical results, together with open problem areas, and outlines the possibilities for further research along with a sketch of possible applications.

Chapter 2
Mathematical Background

The aim of this chapter is to review fundamental terminology concerning formal language theory and related mathematical notation used throughout this thesis and thus to minimize the risk of possible confusion.

This chapter is divided into four sections. Section 2.1 reviews basic notions from set theory and algebra. Section 2.2 gives the survey of definitions of fundamental elements of formal language theory, especially word, language, and operations on them. Finally, the last two sections, Section 2.3 and Section 2.4, summarize essential types of grammars and automata along with their expressive power.

The content of this chapter is based on [29, 66, 88, 92].

2.1 Sets and Relations

A set P is a collection of mutually distinct elements taken from some prespecified universe $\mathbb{U}$. For an element a, $a \in \mathrm{P}$ denotes that a is a member of P. Analogously, $a \notin \mathrm{P}$ denotes that a is not a member of P. The number of members in P, the *cardinality* of P, is denoted by $\mathrm{card}(\mathrm{P})$. P is said to be *finite* if its cardinality is a nonnegative integer. Otherwise, P is said to be *infinite*. The set that has no members, the *empty set*, is denoted by $\emptyset$.

Sets can be characterized by enumerating their elements. For example, $\{a, b, c\}$ characterizes a set with three elements a, b, and c. Whenever the meaning is clear, ellipsis can be used. Thus, a set of all octal digits can be written as $\{0, 1, \ldots, 7\}$. Frequently, a set P can be characterized by stating that P contains all elements from the universe $\mathbb{U}$ that fulfill some property π. This is formally expressed as

$$\mathrm{P} = \{a \mid \pi(a)\}$$

The set of all positive and the set of all nonnegative integers are denoted by $\mathbb{N}$ and $\mathbb{N}_0$, respectively.

For two sets P and Q, $\mathrm{P} \cup \mathrm{Q}$, $\mathrm{P} \cap \mathrm{Q}$, and $\mathrm{P} - \mathrm{Q}$ denote the *union*, *intersection*, and *difference* of sets P and Q, respectively. By $\mathrm{P} \subseteq \mathrm{Q}$, it is denoted that P is a *subset* of Q. Moreover, if $\mathrm{P} \neq \mathrm{Q}$, P is said to be a *proper subset* of Q, written as $\mathrm{P} \subset \mathrm{Q}$. Two sets P and Q are said to be *incomparable* if $\mathrm{P} \nsubseteq \mathrm{Q}$ and $\mathrm{Q} \nsubseteq \mathrm{P}$. The *power set* of P, 2^{P}, is defined as $2^{\mathrm{P}} = \{\mathrm{X} \mid \mathrm{X} \subseteq \mathrm{P}\}$. Sets whose members are other sets are called *families of sets* rather than sets of sets.

Let $n \geq 1$ and let $a_1, a_2, \ldots, a_n$ be n elements. Then $(a_1, a_2, \ldots, a_n)$ denotes the ordered *n-tuple* consisting of n elements $a_1, a_2, \ldots, a_n$ in this order. In further text, terms *pairs*, *triples*, *quadruples*, *quintuples*, *sextuples*, and *septuples* will be used rather than 2-tuples, 3-tuples, 4-tuples, 5-tuples, 6-tuples, and 7-tuples, respectively.

Let n be a positive integer and let $\mathrm{A}_1, \mathrm{A}_2, \ldots, \mathrm{A}_n$ be n sets. The *Cartesian product* of n sets $\mathrm{A}_1, \mathrm{A}_2, \ldots, \mathrm{A}_n$, denoted $\mathrm{A}_1 \times \mathrm{A}_2 \times \cdots \times \mathrm{A}_n$, is defined as a set

$$\mathrm{A}_1 \times \mathrm{A}_2 \times \cdots \times \mathrm{A}_n = \{(x_1, x_2, \ldots, x_n) \mid x_i \in \mathrm{A}_i, 1 \leq i \leq n\}$$

of ordered n-tuples. If $\mathrm{A}_1 = \mathrm{A}_2 = \cdots = \mathrm{A}_n = \mathrm{A}$, the Cartesian product of n sets $\mathrm{A}_1, \mathrm{A}_2, \ldots, \mathrm{A}_n$ will be shortly denoted[4] as $\mathrm{C}_n(\mathrm{A})$.

Let P and Q be two sets and let $\rho \subseteq \mathrm{P} \times \mathrm{Q}$. Then, ρ is said to be a *binary relation*, or just a *relation*[5] for short, from P to Q. Usually, $(p, q) \in \rho$ is written in more convenient way as $p\rho q$. If $\mathrm{P} = \mathrm{Q}$, then ρ is said to be a *relation on* P or a *relation over* P. The *inverse relation* of ρ, ρ^{-1}, is defined as $\rho^{-1} = \{(y, x) \mid x\rho y\}$.

The *domain* of ρ, $\mathrm{domain}(\rho)$, and the *range* of ρ, $\mathrm{range}(\rho)$, are defined as

$$\mathrm{domain}(\rho) = \{x \mid x\rho y, y \in \mathrm{Q}\}$$

and

$$\mathrm{range}(\rho) = \{y \mid x\rho y, x \in \mathrm{P}\}$$

respectively.

For a set P, a relation ρ over P is said to be

(a) *reflexive* if for every $x \in \mathrm{P}$, $x\rho x$;

(b) *antisymmetric* if for every $x, y \in \mathrm{P}$, $x\rho y$ and $y\rho x$ implies $x = y$;

(c) *transitive* if for every $x, y, z \in \mathrm{P}$, $x\rho y$ and $y\rho z$ implies $x\rho z$.

If ρ is a relation over P that is reflexive, antisymmetric, and transitive, then ρ is said to be a *partial order* on P and the pair (P, ρ) is said to be a *partially ordered set*.

Let A be a set. The *identity relation* over A, id_A, is defined as

$$\mathrm{id}_\mathrm{A} = \{(x, x) \mid x \in \mathrm{A}\}$$

For two relations ρ and σ over A,

$$\rho \circ \sigma = \{(x, z) \mid x\rho y, y\sigma z; x, y, z \in \mathrm{A}\}$$

is the *composition* of ρ and σ. For $k \geq 0$, the *k-fold product* of ρ, ρ^k, is recursively defined as follows

$$\rho^k = \begin{cases} \mathrm{id}_\mathrm{A} & \text{for } k = 0 \\ \rho \circ \rho^{k-1} & \text{for } k \geq 1 \end{cases}$$

[4] To not confuse with the nth power of a language L, which is denoted L^n.

[5] In further text, no relations other than binary are used.

The *transitive closure* of ρ, ρ^+, and the *reflexive and transitive closure* of ρ, ρ^*, are defined as

$$\rho^+ = \bigcup_{i=1}^{\infty} \rho^i \qquad \text{and} \qquad \rho^* = \bigcup_{i=0}^{\infty} \rho^i$$

respectively.

Let $(\mathrm{P}, \leq)$ be a partially ordered set and let $\mathrm{S} \subseteq \mathrm{P}$. An element $l \in \mathrm{P}$ is a *minimum*—or least—element in P, denoted $\min \mathrm{P}$, if and only if $p \in \mathrm{P}$ implies $l \leq p$. An element $x \in \mathrm{P}$ is an *upper bound* for S if and only if for every $a \in \mathrm{S}$, $a \leq x$. If x is a least element of all upper bounds of S, then x is called the *supremum* of S, denoted by $\sup \mathrm{S}$.

Similarly, an element $m \in \mathrm{P}$ is a *maximum*—or greatest—element in P, denoted $\max \mathrm{P}$, if and only if $p \in \mathrm{P}$ implies $p \leq m$. An element $x \in \mathrm{P}$ is a *lower bound* for S if and only if for every $a \in \mathrm{S}$, $x \leq a$. If x is a greatest element of all lower bounds of S, then x is called the *infimum* of S, denoted by $\inf \mathrm{S}$.

Let A and B be two sets. A *function* (*mapping*) ϕ from A to B is a relation $\phi \in \mathrm{A} \times \mathrm{B}$ such that for every $a \in \mathrm{A}$

$$\mathrm{card}(\{b \mid a\phi b, b \in \mathrm{B}\}) \leq 1$$

If $\mathrm{domain}(\phi) = \mathrm{A}$, ϕ is said to be *total*. Otherwise, ϕ is *partial*. In case of functions, a preferred way to express that $(x, y) \in \phi$ is $\phi(x) = y$ instead of $x\phi y$.

ϕ is an *injection* if for every $y \in \mathrm{B}$, $\mathrm{card}(\{x \mid \phi(x) = y, x \in \mathrm{A}\}) \leq 1$. ϕ is a *surjection* if for every $y \in \mathrm{B}$, $\mathrm{card}(\{x \mid \phi(x) = y, x \in \mathrm{A}\}) \geq 1$. ϕ is a *bijection* if it is both injection and surjection.

Let S be a finite set and let $\mathrm{I} = \{1, 2, \ldots, \mathrm{card}(\mathrm{S})\}$ be a set of indices. Then, the set of all *permutations* of elements of S, $\mathrm{perm}(\mathrm{S})$, is a set of all bijections from I to S.

Let $n \geq 1$. A set $\mathrm{J} \subseteq \mathrm{C}_n(\mathbb{N}_0)$ is said to be *linear* if there exist $\alpha, \beta_1, \beta_2, \ldots, \beta_m \in \mathrm{C}_n(\mathbb{N}_0)$, $m \geq 0$ such that

$$\mathrm{J} = \{\alpha + \sum_{i=1}^{m} k_i\beta_i \mid k_j \in \mathbb{N}_0, 1 \leq j \leq m\}$$

If J is the union of a finite number of linear sets, then J is said to be *semilinear*.

2.2 Words and Languages

An *alphabet* Σ is a finite nonempty set of elements, called *symbols*. A *word* over Σ is recursively defined as follows: (1) the word that contains no symbols, the *empty word*, denoted by ε, is a word over Σ; (2) for every word w over Σ and for every symbol $a \in \Sigma$, wa is a word over Σ. The set of all words over Σ is denoted as Σ^*. $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ denotes a set of all non-empty words over Σ.

Given a word $x = a_1a_2 \ldots a_k$ from Σ^*, $a_i \in \Sigma$, $1 \leq i \leq k$. Then the *length of* x, $|x|$, is defined as $|x| = k$. For ε, $|\varepsilon| = 0$. If x and y are two words from Σ^*, then xy is the *concatenation* of x and y.

Let $i \geq 0$ and let $x \in \Sigma^*$. The *ith power* of x, x^i, is defined recursively as follows

$$x^i = \begin{cases} \varepsilon & \text{for } i = 0 \\ xx^{i-1} & \text{for } i \geq 1 \end{cases}$$

Let $w \in \Sigma^*$. If $w = uv$ for some $u, v \in \Sigma^*$, then u is said to be a *prefix* of w and v is said to be a *suffix* of w. If $u \notin \{\varepsilon, w\}$, then u is said to be a *proper* prefix of w. Similarly, if $v \notin \{\varepsilon, w\}$, then v is said to be a *proper* suffix of w. If $w = uzv$ for some $u, z, v \in \Sigma^*$, then z is said to be a *subword* of w. If $z \notin \{\varepsilon, w\}$, then z is said to be a *proper* subword of w. $\mathrm{prefix}(w)$, $\mathrm{suffix}(w)$, and $\mathrm{subword}(w)$ denote sets of all prefixes, suffixes, and subwords of w, respectively. The set of all symbols that appear in w is denoted $\mathrm{alph}(w)$, formally

$$\mathrm{alph}(w) = \{a \mid w = uav, a \in \Sigma, u, v \in \Sigma^*\}$$

For $a \in \Sigma$, $\mathscr{O}_a(w)$ denotes the number of occurrences of a in w, formally

$$\mathscr{O}_a(w) = \mathrm{card}(\{u \mid w = uav, u, v \in \Sigma^*\})$$

For $\mathrm{W} \subseteq \Sigma$, the number of occurrences of symbols from W in w, $\mathscr{O}_{\mathrm{W}}(w)$, is defined as

$$\mathscr{O}_{\mathrm{W}}(w) = \sum_{a \in \mathrm{W}} \mathscr{O}_a(w)$$

Let $k \geq 0$. Then

$$\mathrm{kprefix}(w, k) = \begin{cases} u & \text{if } w = uv,\ |u| = k,\ u, v \in \Sigma^* \\ w & \text{otherwise} \end{cases}$$

A *language*, L, over an alphabet Σ is a subset of Σ^*, formally $\mathrm{L} \subseteq \Sigma^*$. If $\mathrm{L} = \Sigma^*$, then L is said to be the *universal language* over Σ.

It is obvious that properties and operations that are applicable on sets are also applicable on languages. Therefore, a language can be *finite*, *infinite*, or *empty*. In addition to set theory operations like *union*, *intersection*, and *difference*, there are also operations that can be performed only with languages. Such an operation is *concatenation* of languages. For two languages L_1 and L_2 over an alphabet Σ, the concatenation of L_1 and L_2, $\mathrm{L}_1\mathrm{L}_2$, is defined as

$$\mathrm{L}_1\mathrm{L}_2 = \{xy \mid x \in \mathrm{L}_1, y \in \mathrm{L}_2\}$$

For some $i \geq 0$, the *ith power* of language L, L^i, is recursively defined as

$$\mathrm{L}^i = \begin{cases} \{\varepsilon\} & \text{for } i = 0 \\ \mathrm{L}\mathrm{L}^{i-1} & \text{for } i \geq 1 \end{cases}$$

The *(Kleene) closure* of L, L^*, and the *positive closure* of L, L^+, are defined as

$$\mathrm{L}^* = \bigcup_{i=0}^{\infty} \mathrm{L}^i \qquad \text{and} \qquad \mathrm{L}^+ = \bigcup_{i=1}^{\infty} \mathrm{L}^i$$

respectively.

Analogously to set theory, sets whose members are languages are called *families of languages*. A language family $\mathscr{L}$ is said to be ε-*free* if for every $\mathrm{L} \in \mathscr{L}$, $\varepsilon \notin \mathrm{L}$.

A language L over an alphabet Σ is said to be a *prefix language* if and only if for every pair of words $u, v \in \mathrm{L}$ it holds that u is not a proper prefix of v. The family of all prefix languages will be denoted as $\mathscr{L}_{\mathrm{pfx}}$.

Let $\Sigma = \{a_1, a_2, \dots, a_n\}$ be an alphabet. For $w \in \Sigma^*$, the *commutative (Parikh) image of* w, $\phi(w)$, is defined as

$$\phi(w) = (\mathscr{O}_{a_1}(w), \mathscr{O}_{a_2}(w), \dots, \mathscr{O}_{a_n}(w))$$

For $\mathrm{L} \subseteq \Sigma^*$, the *commutative (Parikh) map of* L, $\phi(\mathrm{L})$, is defined as

$$\phi(\mathrm{L}) = \{\phi(w) \mid w \in \mathrm{L}\}$$

L is said to be a *semilinear* if and only if $\phi(\mathrm{L})$ is a semilinear set. A language family is semilinear if and only if it contains only semilinear languages.

Let Σ and Δ be two alphabets. A mapping h from Σ^* to Δ^* is called a *morphism* if for every $x, y \in \Sigma^*$ it holds that $h(xy) = h(x)h(y)$. For $\mathrm{L} \subseteq \Sigma^*$, extend the definition of h to $h(\mathrm{L}) = \{h(w) \mid w \in \mathrm{L}\}$.

Let $k \in \mathbb{N}_0$ and let Σ and Δ be two alphabets. For some $\mathrm{L} \subseteq \Sigma^*$, a morphism h from Σ^* to Δ^* is said to be k-*erasing* if and only if $w \in \mathrm{L}$ implies $|w| \leq k|h(w)|$.

Let $\mathscr{L}$ be a family of languages. $\mathscr{L}$ is closed under *linear erasing* if and only if $h(\mathrm{L}) \in \mathscr{L}$ for all $\mathrm{L} \in \mathscr{L}$ and for all k-erasing morphisms h, $k \geq 0$.

2.3 Grammars

This section defines devices for generating languages called *grammars*. In formal language theory, grammars play an important role as language models.

Definition 2.3.1. A *general unrestricted grammar* (abbreviated GUG), G, is a quadruple $\mathrm{G} = (\mathrm{V}, \mathrm{T}, \mathrm{P}, \sigma)$, where

- V is a *total* alphabet;
- $\mathrm{T} \subseteq \mathrm{V}$ is an alphabet of *terminal symbols*;
- $\mathrm{P} \subseteq \mathrm{V}^+ \times \mathrm{V}^*$ is a finite set of *rewriting rules*;
- $\sigma \in \mathrm{V}^+$ is the *axiom*.

Symbols from $(\mathrm{V} - \mathrm{T})$ are called *nonterminal symbols*.

If $\mathrm{P} \subseteq \mathrm{V}^*(\mathrm{V} - \mathrm{T})\mathrm{V}^* \times \mathrm{V}^*$ and $\sigma \in (\mathrm{V} - \mathrm{T})$, then G is said to be an *unrestricted*, or *phrase-structure*, grammar, abbreviated UG. In this case, σ is instead labeled as S and is called the *start symbol*.

Moreover, if $\mathrm{V} = \mathrm{T}$, then G is said to be a *pure grammar* (see [63]), abbreviated PG. In this case, $\mathrm{G} = (\mathrm{V}, \mathrm{T}, \mathrm{P}, \sigma)$ is shortened to $\mathrm{G} = (\mathrm{T}, \mathrm{P}, \sigma)$.

A rewriting rule, or just a *rule* for brevity, $(x, y) \in \mathrm{P}$ is usually written as $x \to y$. Rewriting rules are sometimes called *productions*. A rule $x \to y \in \mathrm{P}$ satisfying $y = \varepsilon$ is said to be an *erasing rule*. If for every rule $x \to y$ from P it holds that $y \neq \varepsilon$, then G is said to be *propagating*.

The G-based relation of *direct derivation*, $\Rightarrow_{\mathrm{G}}$, over V^* is defined as follows: $uxv \Rightarrow_{\mathrm{G}} uyv$ if and only if $x \rightarrow y \in \mathrm{P}$, where $u, v, x, y \in \mathrm{V}^*$. For $k \geq 0$, $\Rightarrow_{\mathrm{G}}^{k}$ denotes the k-fold product of $\Rightarrow_{\mathrm{G}}$; $\Rightarrow_{\mathrm{G}}^{+}$ denotes the transitive closure of $\Rightarrow_{\mathrm{G}}$, and $\Rightarrow_{\mathrm{G}}^{*}$ denotes the reflexive and transitive closure of $\Rightarrow_{\mathrm{G}}$. If $u \Rightarrow_{\mathrm{G}}^{*} v$ for some $u, v \in \mathrm{V}^*$, then $u \Rightarrow_{\mathrm{G}}^{*} v$ is said to be a *derivation* of v from u. When no confusion exists, $u \Rightarrow_{\mathrm{G}}^{*} v$ can be shortened to $u \Rightarrow^{*} v$. A word $x \in \mathrm{V}^*$ is called a *sentential form* if and only if $\sigma \Rightarrow_{\mathrm{G}}^{*} x$.

The *language* generated by G, L(G), is defined as

$$\mathrm{L(G)} = \{w \mid \sigma \Rightarrow_{\mathrm{G}}^{*} w, w \in \mathrm{T}^*\}$$

□

Observe that from the definition of k-fold product, $\Rightarrow_{\mathrm{G}}^{0} = \mathrm{id}_{\mathrm{V}^*}$, and hence $u \Rightarrow_{\mathrm{G}}^{0} v$ if and only if $u = v$.

Sometimes it is useful to label rule $x \rightarrow y$ with a unique label, r, as $r\colon x \rightarrow y$.

Definition 2.3.2. Let $\mathrm{G} = (\mathrm{V}, \mathrm{T}, \mathrm{P}, \sigma)$ be a general unrestricted grammar. Let Ψ be a set of *rule labels* such that $\mathrm{card}(\Psi) = \mathrm{card}(\mathrm{P})$, and ψ be a bijection from Ψ to P that maps labels to rules. For a rule $x \rightarrow y \in \mathrm{P}$ and a label $r \in \Psi$, $r\colon x \rightarrow y$ symbolizes $\psi(r) = x \rightarrow y$. For brevity, to express that $r \in \Psi$, $x \rightarrow y \in \mathrm{P}$, and $\psi(r) = x \rightarrow y$, the notation $r\colon x \rightarrow y \in \mathrm{P}$ is used. In terms of variables and values, Ψ can be considered a set of variables and ψ as evaluation function that maps variables (rule labels) to their values (rules). Rule labels and rules can then be treated equivalently.

For a rule $r : x \rightarrow y \in \mathrm{P}$, x is the *left-hand side* of r, denoted as $\mathrm{lhs}(r)$, and y is the *right-hand side* of r, denoted as $\mathrm{rhs}(r)$.

Let $n \geq 0$. Let $u_0 \Rightarrow_{\mathrm{G}}^{n} u_n$ be a derivation and $\pi_n \in \mathrm{P}^*$ be a sequence of rules such that $u_0, u_n \in \mathrm{V}^*$, $\pi_0 = \varepsilon$ and if $n \geq 1$, then $u_{i-1} \Rightarrow_{\mathrm{G}} u_i$, where $u_{i-1} = z_i\, \mathrm{lhs}(r_i) z_i'$, $u_i = z_i\, \mathrm{rhs}(r_i) z_i'$, $\pi_i = \pi_{i-1} r_i$, $r_i \in \mathrm{P}$, and $z_i, z_i' \in \mathrm{V}^*$, for all $1 \leq i \leq n$. The derivation $u_0 \Rightarrow_{\mathrm{G}}^{n} u_n$ according to π_n is then expressed as $u_0 \Rightarrow_{\mathrm{G}}^{n} u_n\, [\pi_n]$. Depending on the value of n, $u_0 \Rightarrow_{\mathrm{G}}^{n} u_n\, [\pi_n]$ can mean $u_0 \Rightarrow_{\mathrm{G}} u_n\, [\pi_n]$, $u_0 \Rightarrow_{\mathrm{G}}^{*} u_n\, [\pi_n]$, or $u_0 \Rightarrow_{\mathrm{G}}^{+} u_n\, [\pi_n]$. For a word $w \in \mathrm{L(G)}$ such that $\sigma \Rightarrow_{\mathrm{G}}^{*} w\, [\pi]$, π is called a *parse*, or *Szilard word*, of w. □

The family of languages generated by unrestricted grammars coincides with the family of *recursively enumerable* languages, denoted as **RE**. Formally,

$$\mathbf{RE} = \{\mathrm{L(G)} \mid \mathrm{G} \text{ is an unrestricted grammar}\}$$

In connection with recursively enumerable languages, Alonzo Church (see [11]) made the statement later known as *Church's Thesis*. According to Church's Thesis, a language L is recursively enumerable if and only if there exists an effective procedure that characterizes it. Church's Thesis cannot be proven because there is no a formal definition of what should be considered as an effective procedure. In general, the term *effective procedure* is understood as any method for solving problems intuitively considered as algorithmically solvable.

Chomsky Hierarchy of Language Families

The generative capacity of unrestricted grammars can be reduced by restricting the form of the rules. The Chomsky classification of grammars (see [9, 10]) is based on such restrictions.

Definition 2.3.3. Let $\mathrm{G} = (\mathrm{V}, \mathrm{T}, \mathrm{P}, \mathrm{S})$ be an unrestricted grammar.

- If every rule $x \to y \in (\mathrm{P} - \{\mathrm{S} \to \varepsilon\})$ is of the form
  $$x = u\mathrm{A}v, y = uzv$$
  where $\mathrm{A} \in (\mathrm{V} - \mathrm{T})$, $z \in \mathrm{V}^+$, $u, v \in \mathrm{V}^*$, and $\mathrm{S} \to \varepsilon \in \mathrm{P}$ implies that $\mathrm{S} \notin \mathrm{alph}(y)$, then G is said to be a *context-sensitive grammar* (abbreviated CSG). The language generated by context-sensitive grammar is called a *context-sensitive* language. The family of context-sensitive languages is denoted as **CS**.
- If every rule from P is of the form $\mathrm{A} \to x$, where $\mathrm{A} \in (\mathrm{V} - \mathrm{T})$, $x \in \mathrm{V}^*$, then G is said to be a *context-free grammar* (abbreviated CFG). The language generated by context-free grammar is called a *context-free* language. The family of context-free languages is denoted as **CF**.
- If every rule from P is of the form $\mathrm{A} \to x\mathrm{B}y$ or $\mathrm{A} \to x$, where $\mathrm{A}, \mathrm{B} \in (\mathrm{V} - \mathrm{T})$, $x, y \in \mathrm{T}^*$, then G is said to be a *linear grammar* (abbreviated LG). The language generated by linear grammar is called a *linear* language. The family of linear languages is denoted as **LIN**.
- If every rule from P is of the form $\mathrm{A} \to x\mathrm{B}$ or $\mathrm{A} \to x$, where $\mathrm{A}, \mathrm{B} \in (\mathrm{V} - \mathrm{T})$, $x \in \mathrm{T}^*$, then G is said to be a *right-linear grammar* (abbreviated RLG). Moreover, if every rule from $(\mathrm{P} - \{\mathrm{S} \to \varepsilon\})$ is of the form $\mathrm{A} \to a\mathrm{B}$ or $\mathrm{A} \to a$, where $\mathrm{A}, \mathrm{B} \in (\mathrm{V} - \mathrm{T})$, $a \in \mathrm{T}$, and $\mathrm{S} \to \varepsilon \in \mathrm{P}$ implies that $\mathrm{B} \neq \mathrm{S}$, then G is said to be a *regular grammar* (abbreviated RG). The language generated by a right-linear and a regular grammar is called a *right-linear* and a *regular* language, respectively. The family of right-linear languages and the family of regular languages coincide and are denoted as **REG**.

□

The following theorem represents the Chomsky hierarchy of language families.

Theorem 2.3.4.
$$\mathbf{REG} \subset \mathbf{LIN} \subset \mathbf{CF} \subset \mathbf{CS} \subset \mathbf{RE}$$

Proof. See [66]. □

An important subfamily of context-free languages are the *Dyck languages*. Let $n \geq 1$ and $\Sigma_n = \{a_i, b_i \mid 1 \leq i \leq n\}$. The *Dyck language* $\mathscr{D}_n$ over Σ_n is generated by the grammar
$$(\{\mathrm{S}\} \cup \Sigma_n, \Sigma_n, \{\mathrm{S} \to \mathrm{SS}, \mathrm{S} \to \varepsilon\} \cup \{\mathrm{S} \to a_i\mathrm{S}b_i \mid 1 \leq i \leq n\}, \mathrm{S})$$

2.4 Automata

While grammars work as language generators, automata work as language recognizers. In general, given a word, w, and an automaton, M, w belongs to the language recognized by M if and only if M enters the final configuration after reading all symbols of w. This section recalls definitions of several fundamental types of automata.

Definition 2.4.1. A *finite automaton* (abbreviated FA), M, is a quintuple $\mathrm{M} = (\mathrm{Q}, \Sigma, \mathrm{R}, s, \mathrm{F})$, where

- Q is a finite set of *states*;
- Σ is an *input alphabet*, $\mathrm{Q} \cap \Sigma = \emptyset$;
- $\mathrm{R} \subseteq \mathrm{Q}(\Sigma \cup \{\varepsilon\}) \times \mathrm{Q}$ is a finite set of *rules*;
- $s \in \mathrm{Q}$ is the *initial state*;
- $\mathrm{F} \subseteq \mathrm{Q}$ is a set of *final states*.

A rule $(pa, q) \in \mathrm{R}$ is usually written as $pa \to q$. M is said to be a *deterministic* finite automaton (abbreviated dFA) if and only if $\mathrm{R} \subseteq \mathrm{Q}\Sigma \times \mathrm{Q}$ and for every rule $r \in \mathrm{R}$, it holds

$$\mathrm{card}(\{r' \mid r' \in \mathrm{R}, \mathrm{lhs}(r') = \mathrm{lhs}(r)\}) \leq 1$$

A *configuration* of M is a word from $\mathrm{Q}\Sigma^*$.

The M-based relation of *direct move*, $\vdash_{\mathrm{M}}$, over $\mathrm{Q}\Sigma^*$ is defined as follows: $paw \vdash_{\mathrm{M}} qw$ if and only if $pa \to q \in \mathrm{R}$, $p, q \in \mathrm{Q}$, $a \in (\Sigma \cup \{\varepsilon\})$, and $w \in \Sigma^*$. As usual, for $k \geq 0$, $\vdash_{\mathrm{M}}^k$ denotes the k-fold product of $\vdash_{\mathrm{M}}$; $\vdash_{\mathrm{M}}^*$ denotes the reflexive and transitive closure of $\vdash_{\mathrm{M}}$, and $\vdash_{\mathrm{M}}^+$ denotes the transitive closure of $\vdash_{\mathrm{M}}$. If $\chi \vdash_{\mathrm{M}}^* \chi'$ for some configurations χ and χ' of M, then $\chi \vdash_{\mathrm{M}}^* \chi'$ is said to be a *move* from χ to χ'. When no confusion exists, $\chi \vdash_{\mathrm{M}}^* \chi'$ can be shortened to $\chi \vdash^* \chi'$.

The *language accepted* by M, L(M), is defined as

$$\mathrm{L}(\mathrm{M}) = \{w \mid sw \vdash_{\mathrm{M}}^* f, w \in \Sigma^*, f \in \mathrm{F}\}$$

$\mathscr{L}(\mathrm{FA})$ and $\mathscr{L}(\mathrm{dFA})$ denote the family of languages accepted by finite automata and the family of languages accepted by deterministic finite automata, respectively. □

Notions defined in Definition 2.3.2 can also be extended to finite and other kinds of automata. Thus, if M is an arbitrary automaton, $n \in \mathbb{N}_0$, and χ_0 and χ_n are two configurations of M, then $\chi_0 \vdash_{\mathrm{M}}^n \chi_n\ [\pi_n]$ denotes the move from χ_0 to χ_n according to π_n. If w is a word over M's input alphabet that is contained in χ_0 and χ_n is a final configuration of M, then π_n is a *parse* of w.

Theorem 2.4.2.

$$\mathscr{L}(\mathrm{FA}) = \mathscr{L}(\mathrm{dFA}) = \mathbf{REG}$$

Proof. See [66]. □

Definition 2.4.3. A *pushdown automaton* (abbreviated PDA), M, is a septuple $M = (Q, \Sigma, \Gamma, R, s, S, F)$, where

- Q is a finite set of *states*;
- Σ is an *input alphabet*;
- Γ is a *pushdown alphabet*, Q, Σ and Γ are pairwise disjoint;
- $R \subseteq \Gamma Q(\Sigma \cup \{\varepsilon\}) \times \Gamma^* Q$ is a finite set of *rules*;
- $s \in Q$ is the *initial state*;
- $S \in \Gamma$ is the *initial pushdown symbol*;
- $F \subseteq Q$ is a set of *final states*.

A rule $(Apa, xq) \in R$ is usually written as $Apa \to xq$. M is said to be a *deterministic* pushdown automaton (abbreviated dPDA) if and only if for every rule $r \in R$, it holds that

$$\mathrm{card}(\{r' \mid r' \in R, \mathrm{lhs}(r') \in \mathrm{prefix}(\mathrm{lhs}(r))\}) \leq 1$$

A *configuration* of M is a word from $\Gamma^* Q \Sigma^*$.

The M-based relation of *direct move*, $\vdash_M$, over $\Gamma^* Q \Sigma^*$ is defined as follows: $uApaw \vdash_M uxqw$ if and only if $Apa \to xq \in R$, $A \in \Gamma$, $u, x \in \Gamma^*$, $p, q \in Q$, $a \in (\Sigma \cup \{\varepsilon\})$, and $w \in \Sigma^*$. As usual, for $k \geq 0$, $\vdash_M^k$ denotes the k-fold product of $\vdash_M$; $\vdash_M^*$ denotes the reflexive and transitive closure of $\vdash_M$, and $\vdash_M^+$ denotes the transitive closure of $\vdash_M$. If $\chi \vdash_M^* \chi'$ for some configurations χ and χ' of M, then $\chi \vdash_M^* \chi'$ is said to be a *move* from χ to χ'. When no confusion exists, $\chi \vdash_M^* \chi'$ can be shortened to $\chi \vdash^* \chi'$.

There are three types of languages accepted by M:

(a) the language accepted by M by *final state*, $L(M)_f$, defined as

$$L(M)_f = \{w \mid Ssw \vdash_M^* \gamma f, w \in \Sigma^*, \gamma \in \Gamma^*, f \in F\}$$

(b) the language accepted by M by *empty pushdown*, $L(M)_\varepsilon$, defined as

$$L(M)_\varepsilon = \{w \mid Ssw \vdash_M^* q, w \in \Sigma^*, q \in Q\}$$

(c) the language accepted by M by *final state and empty pushdown*, $L(M)_{f\varepsilon}$, defined as

$$L(M)_{f\varepsilon} = \{w \mid Ssw \vdash_M^* f, w \in \Sigma^*, f \in F\}$$

$\mathscr{L}(\mathrm{PDA})_f$, $\mathscr{L}(\mathrm{PDA})_\varepsilon$, and $\mathscr{L}(\mathrm{PDA})_{f\varepsilon}$ denote the family of languages accepted by pushdown automata by final state, the family of languages accepted by pushdown automata by empty pushdown, and the family of languages accepted by pushdown automata by final state and empty pushdown, respectively. Analogously, $\mathscr{L}(\text{dPDA})_f$, $\mathscr{L}(\text{dPDA})_\varepsilon$, and $\mathscr{L}(\text{dPDA})_{f\varepsilon}$ denote the family of languages accepted by deterministic pushdown automata by final state, the family of languages accepted by deterministic pushdown automata by empty pushdown, and the family of languages accepted by deterministic pushdown automata by final state and empty pushdown, respectively. □

Theorem 2.4.4. *Following identities and inclusions hold:*

(a) $\mathscr{L}(\text{PDA})_f = \mathscr{L}(\text{PDA})_\varepsilon = \mathscr{L}(\text{PDA})_{f\varepsilon} = \mathbf{CF}$

(b) $\mathbf{REG} \subset \mathscr{L}(\text{dPDA})_f \subset \mathbf{CF}$

(c) $\mathscr{L}(\text{dPDA})_\varepsilon = \mathscr{L}(\text{dPDA})_{f\varepsilon} = (\mathscr{L}(\text{dPDA})_f \cap \mathscr{L}_{\text{pfx}})$

Proof. See [66, 88]. □

The $\mathscr{L}(\text{dPDA})_f$ language family coincides with the family of languages called *deterministic context-free languages*. The $(\mathscr{L}(\text{dPDA})_f \cap \mathscr{L}_{\text{pfx}})$ language family is called the family of *deterministic-prefix context-free languages*. Observe that contrary to PDA, $\mathscr{L}(\text{dPDA})_\varepsilon \subset \mathscr{L}(\text{dPDA})_f$. To demonstrate this, consider a deterministic context-free language L that is not a prefix language, e.g. $\text{L} = \{a, aa\}$, and a dPDA M such that $\text{L(M)}_f = \text{L}$. Clearly, there is no way how to convert M to dPDA M′ such that $\text{L(M)}_\varepsilon = \text{L}$ because when M′ reads a, it is not possible to decide whether to read the next symbol a or empty the pushdown in a deterministic way.

Definition 2.4.5. Let $\text{M} = (\text{Q}, \Sigma, \Gamma, \text{R}, s, \text{S}, \text{F})$ be a PDA. For $q_1, q_2, q_3 \in \text{Q}$, $\gamma_1, \gamma_2, \gamma_3 \in \Gamma^*$, and $w_1, w_2, w_3 \in \Sigma^*$, a move $\gamma_1 q_1 w_1 \vdash \gamma_2 q_2 w_2 \vdash \gamma_3 q_3 w_3$ is said to be a *turn* if $|\gamma_1| \leq |\gamma_2|$ and $|\gamma_2| > |\gamma_3|$. Let $\alpha, \beta, \chi, \chi'$ be four configurations of M and let $\alpha \vdash^* \beta$ and $\chi \vdash^* \chi'$ be two moves. Then $\chi \vdash^* \chi'$ is said to be *included*, or *contained*, in $\alpha \vdash^* \beta$ if $\alpha \vdash^* \chi \vdash^* \chi' \vdash^* \beta$. M is said to be a *one-turn* PDA (abbreviated 1t-PDA) if for all $w \in \Sigma^*$, every move from $\text{S}sw$ includes at most one turn. Analogously to PDA, $\mathscr{L}(\text{1t-PDA})_f$, $\mathscr{L}(\text{1t-PDA})_\varepsilon$, and $\mathscr{L}(\text{1t-PDA})_{f\varepsilon}$ denote language families accepted by one-turn PDA by final state, empty pushdown, and by final state and empty pushdown, respectively. □

Theorem 2.4.6. *Let* $\text{X} \in \{f, \varepsilon, f\varepsilon\}$. *Then,* $\mathscr{L}(\text{1t-PDA})_\text{X} = \mathbf{LIN}$.

Proof. See [88]. □

Definition 2.4.7. A *two-pushdown automaton* (abbreviated 2PDA), M, is an 8-tuple $\text{M} = (\text{Q}, \Sigma, \Gamma, \text{R}, s, \text{S}_1, \text{S}_2, \text{F})$, where

- Q is a finite set of *states*;
- Σ is an *input alphabet*;
- Γ is a *pushdown alphabet*, Q, Σ, and Γ are pairwise disjoint;

- $\mathrm{R} \subseteq \Gamma\{\#\}\Gamma\mathrm{Q}(\Sigma \cup \{\varepsilon\}) \times \Gamma^*\{\#\}\Gamma^*\mathrm{Q}$ is a finite set of *rules*, $\# \notin (\mathrm{Q} \cup \Sigma \cup \Gamma)$;
- $s \in \mathrm{Q}$ is the *initial state*;
- $\mathrm{S}_1 \in \Gamma$ is the *initial symbol on pushdown 1*;
- $\mathrm{S}_2 \in \Gamma$ is the *initial symbol on pushdown 2*;
- $\mathrm{F} \subseteq \mathrm{Q}$ is a set of *final states*.

A rule $(\mathrm{A\#B}pa, x\#yq) \in \mathrm{R}$ is usually written as $\mathrm{A\#B}pa \rightarrow x\#yq$. M is said to be a *deterministic* two-pushdown automaton (abbreviated d2PDA) if and only if for every rule $r \in \mathrm{R}$, it holds that

$$\mathrm{card}(\{r' \mid r' \in \mathrm{R}, \mathrm{lhs}(r') \in \mathrm{prefix}(\mathrm{lhs}(r))\}) \leq 1$$

A *configuration* of M is a word from $\Gamma^*\{\#\}\Gamma^*\mathrm{Q}\Sigma^*$.

The M-based relation of *direct move*, $\vdash_{\mathrm{M}}$, over $\Gamma^*\{\#\}\Gamma^*\mathrm{Q}\Sigma^*$ is defined as follows:

$$u\mathrm{A}\#v\mathrm{B}paw \vdash_{\mathrm{M}} ux\#vyqw \quad \text{if and only if} \quad \mathrm{A\#B}pa \rightarrow x\#yq \in \mathrm{R}$$

where $\mathrm{A}, \mathrm{B} \in \Gamma$, $u, v, x, y \in \Gamma^*$, $p, q \in \mathrm{Q}$, $a \in (\Sigma \cup \{\varepsilon\})$, and $w \in \Sigma^*$. As usual, for $k \geq 0$, $\vdash_{\mathrm{M}}^{k}$ denotes the k-fold product of $\vdash_{\mathrm{M}}$; $\vdash_{\mathrm{M}}^{*}$ denotes the reflexive and transitive closure of $\vdash_{\mathrm{M}}$, and $\vdash_{\mathrm{M}}^{+}$ denotes the transitive closure of $\vdash_{\mathrm{M}}$. If $\chi \vdash_{\mathrm{M}}^{*} \chi'$ for some configurations χ and χ' of M, then $\chi \vdash_{\mathrm{M}}^{*} \chi'$ is said to be a *move* from χ to χ'. When no confusion exists, $\chi \vdash_{\mathrm{M}}^{*} \chi'$ can be shortened to $\chi \vdash^{*} \chi'$.

There are three types of languages accepted by M:

(a) the language accepted by M by *final state*, $\mathrm{L(M)}_f$, defined as

$$\mathrm{L(M)}_f = \{w \mid \mathrm{S}_1\#\mathrm{S}_2sw \vdash_{\mathrm{M}}^{*} \gamma_1\#\gamma_2 f, w \in \Sigma^*, \gamma_1, \gamma_2 \in \Gamma^*, f \in \mathrm{F}\}$$

(b) the language accepted by M by *empty pushdown*, $\mathrm{L(M)}_\varepsilon$, defined as

$$\mathrm{L(M)}_\varepsilon = \{w \mid \mathrm{S}_1\#\mathrm{S}_2sw \vdash_{\mathrm{M}}^{*} \#q, w \in \Sigma^*, q \in \mathrm{Q}\}$$

(c) the language accepted by M by *final state and empty pushdown*, $\mathrm{L(M)}_{f\varepsilon}$, defined as

$$\mathrm{L(M)}_{f\varepsilon} = \{w \mid \mathrm{S}_1\#\mathrm{S}_2sw \vdash_{\mathrm{M}}^{*} \#f, w \in \Sigma^*, f \in \mathrm{F}\}$$

$\mathscr{L}(\mathrm{2PDA})_f$, $\mathscr{L}(\mathrm{2PDA})_\varepsilon$, and $\mathscr{L}(\mathrm{2PDA})_{f\varepsilon}$ denote the family of languages accepted by two-pushdown automata by final state, the family of languages accepted by two-pushdown automata by empty pushdown, and the family of languages accepted by two-pushdown automata by final state and empty pushdown, respectively. Analogously, $\mathscr{L}(\mathrm{d2PDA})_f$, $\mathscr{L}(\mathrm{d2PDA})_\varepsilon$, and $\mathscr{L}(\mathrm{d2PDA})_{f\varepsilon}$ denote the family of languages accepted by deterministic two-pushdown automata by final state, the family of languages accepted by deterministic two-pushdown automata by empty pushdown, and the family of languages accepted by deterministic two-pushdown automata by final state and empty pushdown, respectively. □

Definition 2.4.8. Let $M = (Q, \Sigma, \Gamma, R, s, S_1, S_2, F)$ be a two-pushdown automaton. For $x_1, x_2, y_1, y_2, z_1, z_2 \in \Gamma^*$, $o, p, q \in Q$, $a, b \in (\Sigma \cup \{\varepsilon\})$, and $w \in \Sigma^*$, let T denote a move

$$x_1\#x_2oabw \vdash y_1\#y_2pbw \vdash z_1\#z_2qw$$

If for some $i \in \{1, 2\}$, it holds that $|x_i| \leq |y_i|$ and $|y_i| > |z_i|$, then T is said to be a *turn*. If for all $i \in \{1, 2\}$, it holds that $|x_i| \leq |y_i|$ and $|y_i| > |z_i|$, then T is said to be a *simultaneous turn*. M is said to be a *simultaneously one-turn* 2PDA (abbreviated s1t-2PDA) if for all $w \in \Sigma^*$, every move from $S_1\#S_2sw$ either includes no turn or includes one simultaneous turn. Analogously to 2PDA, $\mathscr{L}(\text{s1t-2PDA})_f$, $\mathscr{L}(\text{s1t-2PDA})_\varepsilon$, and $\mathscr{L}(\text{s1t-2PDA})_{f\varepsilon}$ denote language families accepted by simultaneously one-turn 2PDA by final state, empty pushdown, and by final state and empty pushdown, respectively. □

Theorem 2.4.9. *Let* $X \in \{\text{2PDA, d2PDA, s1t-2PDA}\}$ *and* $Y \in \{f, \varepsilon, f\varepsilon\}$. *Then,* $\mathscr{L}(X)_Y = \mathbf{RE}$.

Proof. See [66, 67]. □

Last, recall a definition of Turing machine and linear bounded automaton that will be used in proofs appearing later in this thesis.

Definition 2.4.10. A *Turing machine* (abbreviated TM), M, is a septuple

$$M = (Q, \Sigma, \Gamma, \Delta, R, s, F)$$

where

- Q is a finite set of *states*;
- Σ is an *input alphabet*;
- Γ is a *tape alphabet*, $\Gamma \cap Q = \emptyset$, $\Sigma \subset \Gamma$;
- $\Delta \in (\Gamma - \Sigma)$ is the *blank symbol*;
- $R \subseteq Q \times \Gamma \times Q \times \Gamma \times \{d_L, d_R\}$ is a finite set of *rules*;
- $s \in Q$ is the *initial state*;
- $F \subseteq Q$ is the set of *final states*.

A rule $(p, X, q, Y, d) \in R$ is usually written as $pX \rightarrow qYd \in R$.

A *configuration* of M is a word from $\Gamma^*Q\Gamma^+$.

The M-based relation of *direct move*, $\vdash_{\mathrm{M}}$, over $\Gamma^*\mathrm{Q}\Gamma^+$ is defined as follows:

1. $\alpha p\mathrm{X}\beta \vdash_{\mathrm{M}} \alpha\mathrm{Y}q\beta$ if and only if $p\mathrm{X} \to q\mathrm{Y}d_{\mathrm{R}} \in \mathrm{R}$
2. $\alpha p \vdash_{\mathrm{M}} \alpha\mathrm{Y}q$ if and only if $p\Delta \to q\mathrm{Y}d_{\mathrm{R}} \in \mathrm{R}$
3. $\alpha\mathrm{Z}p\mathrm{X}\beta \vdash_{\mathrm{M}} \alpha q\mathrm{Z}\mathrm{Y}\beta$ if and only if $p\mathrm{X} \to q\mathrm{Y}d_{\mathrm{L}} \in \mathrm{R}$
4. $\alpha\mathrm{Z}p \vdash_{\mathrm{M}} \alpha q\mathrm{Z}\mathrm{Y}$ if and only if $p\Delta \to q\mathrm{Y}d_{\mathrm{L}} \in \mathrm{R}$

where $p, q \in \mathrm{Q}$, $\mathrm{X}, \mathrm{Y}, \mathrm{Z} \in \Gamma$, and $\alpha, \beta \in \Gamma^*$. For $k \geq 0$, $\vdash_{\mathrm{M}}^{k}$, $\vdash_{\mathrm{M}}^{+}$, and $\vdash_{\mathrm{M}}^{*}$ are defined as usual.

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

$$\mathrm{L}(\mathrm{M}) = \{w \mid sw \vdash_{\mathrm{M}}^{*} \alpha f\beta, w \in \Sigma^*, \alpha, \beta \in \Gamma^*, f \in \mathrm{F}\}$$

If for every $w \in \Sigma^*$, $sw \vdash_{\mathrm{M}}^{*} \chi$ implies that $|sw| \geq |\chi|$, where $\chi \in \Gamma^*\mathrm{Q}\Gamma^+$, M is called a *linear bounded automaton* (abbreviated LBA).

Languages accepted by Turing machines and linear bounded automata are exactly recursively enumerable and context-sensitive languages, respectively. □

Chapter 3
Rewriting Systems as Language Models

Formally, a rewriting system (see [90]) is a pair (V, R), where V is a total alphabet and $\mathrm{R} \subseteq \mathrm{V}^* \times \mathrm{V}^*$ is a finite set of rewriting rules. The definition of the rewriting system is so general to cover any language model based on rewriting. Thus, grammars, automata, and even grammar and automata systems can be all considered to be rewriting systems. Given a rewriting system $\Omega = (\mathrm{V}, \mathrm{R})$, it depends only on the meaning of symbols from V and the definition of a rewriting step and the language of Ω whether Ω represents a grammar or an automaton. For example, Definition 3.1 in [90] on page 46 defines finite deterministic automaton as a rewriting system (Σ, P), where Σ is divided into two disjoint alphabets: alphabet Σ_{Q} of states and alphabet Σ_{T} of input symbols. However, it is more convenient to have these alphabets defined separately, as demonstrated in Definition 2.4.1.

The aim of this chapter is to give a survey[6] of rewriting systems known so far that were studied during the research of new language models presented in the second part of this thesis.

Following the introductory Chapter 1, this chapter is organized into two sections. Section 3.1 is dedicated to regulated rewriting systems. Here are discussed selected types of regulated grammars and automata, like programmed grammars, deep pushdown automata, and jumping grammars. Section 3.2 is focused on rewriting systems where parallel and cooperative rewriting is performed. This includes for instance L-systems and various types of grammar and automata systems. In both sections, attention is paid to how rewriting is done and how it influences the expressive power of a given rewriting system.

3.1 Regulated Rewriting

Given a rewriting system $\Omega = (\mathrm{V}, \mathrm{R})$ and a word $u \in \mathrm{V}^*$. In general, every rule $x \rightarrow y \in \mathrm{P}$, where x has an occurrence in u, can be applied on u. In case that Ω is regulated, there is an additional regulating device (see [21]) that chooses from all possible applicable rules only a subset of candidates that fulfill given conditions.

In [21], regulated rewriting is classified into three groups:

- prescribed sequences of rules;

[6] Grammars and automata that have a direct connection with the Chomsky hierarchy form a fundamental part of the classical theory of formal languages and hence they were discussed in Chapter 2 instead of here. This chapter is primarily focused on rewriting systems from modern formal language theory.

- dependence on previously applied rules;
- context conditions.

Meduna and Zemek (see [72]) divide regulated rewriting into rule-based and context-based, where rule-based regulated rewriting contains the first two groups from the classification in [21]. Note that the classification of regulated rewriting is not comprehensive and also that there are no strict borders between these groups. There are also cases with combined regulations (see Chapter 6 in [21]).

Regulated Grammars

As noted in the introduction, regulated grammars were introduced as a consequence of attempts to increase the generative capacity of context-free grammars. The early idea of how to do it was controlling derivation steps using *matrices*.

Such regulated grammars are called *matrix grammars* (see [1]). A matrix grammar H consists of a context-free grammar $G = (V, T, P, S)$ and a finite set of matrices M, where matrix is a sequence of rules from P. A word $w \in L(G)$ belongs to L(H) if and only if its parse can be broken to matrices from M.

As shown in Example 1.1.1 in [21], there exists a matrix grammar that can generate the non-context-free language $\{a^n b^n c^n \mid n \geq 1\}$. On the other hand, as it follows from Theorem 2.2 and Theorem 2.1 from Chapter 3 in [87], there are context-sensitive languages that cannot be generated by any matrix grammar.

A similar approach as in matrix grammars can be seen also in *regular-controlled grammars* (see [5]). Like matrix grammars, also a regular-controlled grammar H consists of a context-free grammar G, but instead of a finite set of matrices it uses as its regulating device a regular language Ψ, called *control language*, over the set of rules of G. A word $w \in L(G)$ belongs to L(H) if and only if its parse belongs to Ψ.

Matrix grammars and regular-controlled grammars are examples of grammars regulated by prescribed sequences of rules. An example of regulated grammars where rule to be applied depends on rules that were applied previously are *programmed grammars*.

Definition 3.1.1. A *programmed grammar with appearance checking* (see [82]), abbreviated PRG, is a triple $H = (G, \sigma, \phi)$, where

- $G = (V, T, P, S)$ is a context-free grammar;
- σ and ϕ are total mappings from P to 2^P.

Let $r \in P$. Then, $\sigma(r)$ and $\phi(r)$ are said to be the *success field* of r and the *failure field* of r, respectively. If G is propagating, then H is said to be a *propagating programmed grammar with appearance checking*. If for every $p\colon A \to x \in P$ it is satisfied that $\phi(p) = \emptyset$, then H is said to be a *programmed grammar without appearance checking* or just *programmed grammar*.

The H-based relation of *direct derivation*, $\Rightarrow_{\mathrm{H}}$, over $(\mathrm{V}^* \times \mathrm{P})$ is defined as follows: $(y, p) \Rightarrow_{\mathrm{H}} (z, r)$ if and only if either

$$y = u\mathrm{A}v, z = uxv, p\colon \mathrm{A} \to x \in \mathrm{P}, r \in \sigma(p)$$

or

$$y = z, p\colon \mathrm{A} \to x \in \mathrm{P}, \mathrm{A} \notin \mathrm{alph}(y), r \in \phi(p)$$

where $\mathrm{A} \in (\mathrm{V} - \mathrm{T})$ and $u, v, x \in \mathrm{V}^*$. As usual, for $k \geq 0$, $\Rightarrow_{\mathrm{H}}^{k}$ denotes the k-fold product of $\Rightarrow_{\mathrm{H}}$, $\Rightarrow_{\mathrm{H}}^{+}$ denotes the transitive closure of $\Rightarrow_{\mathrm{H}}$, and $\Rightarrow_{\mathrm{H}}^{*}$ denotes the reflexive and transitive closure of $\Rightarrow_{\mathrm{H}}$. If $\alpha \Rightarrow_{\mathrm{H}}^{*} \beta$ for some $\alpha, \beta \in (\mathrm{V}^* \times \mathrm{P})$, then $\alpha \Rightarrow_{\mathrm{H}}^{*} \beta$ is said to be a *derivation* of β from α. Whenever it is clear, $\alpha \Rightarrow_{\mathrm{H}}^{*} \beta$ can be briefly written as $\alpha \Rightarrow^{*} \beta$.

The language generated by H, L(H), is defined as

$$\mathrm{L}(\mathrm{H}) = \{w \mid (\mathrm{S}, p) \Rightarrow_{\mathrm{H}}^{*} (w, r), w \in \mathrm{T}^*, p, r \in \mathrm{P}\}$$

$\mathbf{PRG}$, $\mathbf{PRG}^{-\varepsilon}$, $\mathbf{PRG}_{\mathrm{ac}}$, and $\mathbf{PRG}_{\mathrm{ac}}^{-\varepsilon}$ denote the family of languages generated by programmed grammars, the family of languages generated by propagating programmed grammars, the family of languages generated by programmed grammars with appearance checking, and the family of languages generated by propagating programmed grammars with appearance checking, respectively. □

Since programmed grammars, matrix grammars and regular-controlled grammars are mutually convertible (see Theorem 2.2 and Theorem 2.4 from Chapter 3 in [87]), they characterize the same family of languages.

Observe that programmed grammars with appearance checking have a method to handle situations when a rule cannot be applied. This can be extended also to regular-controlled grammars (see [27]) and matrix grammars (see [1]) by adding an *appearance checking set*. In short, if a context-free rule cannot be applied and its left-hand side appears in appearance checking set, its application is skipped and a derivation proceeds with a next rule.

Appearance checking provides a framework to do simple branching based on symbol appearance to matrix, regular-controlled, and programmed grammars. For instance, one can implement constructions like *"if* X *has at least one occurrence in sentential form, apply rule* r_1*; otherwise, apply rule* r_2*"* with it. Such an extension is powerful enough to describe arbitrary recursively enumerable language (see Theorem 1.2.5 and Theorem 2.1.1' in [21]).

As an example of regulated grammars with regulation based on context conditions is worth mentioning *random context grammars* (see [95]). In a random context grammar, every context-free rule has two sets associated with it—a set of permitting symbols and a set of forbidding symbols. A rule can be then applied on the sentential form if and only if all its permitting symbols occur in the sentential form and simultaneously none from its forbidding symbols occurs in it. If the set of forbidding (permitting) symbols is empty for every rule in a random context grammar G, then G is said to be a *permitting (forbidding) grammar*. In their general form, random context grammars are capable of generating every recursively enumerable language (Theorem 2.7, Chapter 3 in [87]). In [98] has been proven that permitting grammars and propagating permitting grammars generate the same family of languages. The question of whether the same holds also for forbidding grammars remains open.

Besides matrix, programmed, regular-controlled, and random context grammars many other types of regulated grammars can be found in the literature (see [20, 21, 72, 87]). Some examples worth to mention are *vector grammars*[12], *one-sided random context grammars*[70], *indexed grammars*[2], *tree controlled grammars*[19], *additive/multiplicative valence grammars*[76], *ordered grammars*[25], or *macro grammars*[24].

Finite Index

Roughly speaking, the *index* of a grammar G is the number that limits the number of occurrences of nonterminal symbols in a sentential form of G. If such a number is a positive integer, then the index of grammar is *finite*. Otherwise, it is *infinite*. The following definition recalls the (slightly modified) definition of the index of grammar from [21].

Definition 3.1.2. Let G be an arbitrary grammar and let V, T, P, and S be its total alphabet, alphabet of terminal symbols, set of rewriting rules, and start symbol, respectively. For $\pi \in P^*$, define

$$\mathrm{Ind}(\pi, G) = \max\{\mathscr{O}_{V-T}(x) \mid S \Rightarrow_G^* x\,[\rho], x \in V^*, \rho \in \mathrm{prefix}(\pi)\}$$

Furthermore, for $w \in T^*$, define

$$\mathrm{Ind}(w, G) = \min\{\mathrm{Ind}(\pi, G) \mid S \Rightarrow_G^* w\,[\pi]\}$$

Then, the *index of* G, Ind(G), is defined as

$$\mathrm{Ind}(G) = \sup\{\mathrm{Ind}(w, G) \mid w \in L(G)\}$$

Let **X** be the family of languages generated by grammars of type X and let $\mathscr{G}(X)$ be a set of grammars of type X. Then, for $L \in \mathbf{X}$, define

$$\mathrm{Ind}_X(L) = \inf\{\mathrm{Ind}(G) \mid L(G) = L, G \in \mathscr{G}(X)\}$$

For a family **X**, set

$$\begin{aligned} {}_n\mathbf{X} &= \{L \mid L \in \mathbf{X}, \mathrm{Ind}_X(L) \leq n\}, n \geq 1 \\ {}_{\mathrm{fin}}\mathbf{X} &= \bigcup_{n \geq 1} {}_n\mathbf{X} \end{aligned}$$

□

With the finite index restriction introduced, all language families of regulated grammars discussed so far coincide and even appearance checking has no effect here. This is due to the fact that the order of appearance of nonterminal symbols, together with their number of occurrences, can be encoded in a finite number of ways. As a result, regulated grammar of one type is able to simulate a regulated grammar of a different type. However, this does not apply to all regulated grammars, see for instance Theorem 3.1.4 and Theorem 3.1.5 in [21]. Another interesting fact about the finite index is that its introduction leads to an infinite hierarchy of language families. The following theorem demonstrates this on programmed grammars but it is also present in matrix grammars, random context grammars, and other types of regulated grammars (see Chapter 3 in [21]).

Theorem 3.1.3. *Let* $\mathbf{X}$ *be one of* $\mathbf{PRG}$, $\mathbf{PRG}^{-\varepsilon}$, $\mathbf{PRG}_{\mathrm{ac}}$, *or* $\mathbf{PRG}_{\mathrm{ac}}^{-\varepsilon}$. *Let* $n \geq 1$. *Then,* ${}_{n}\mathbf{X} \subset {}_{n+1}\mathbf{X}$.

Proof. See [21]. □

The proof of Theorem 3.1.3 is based on pumping lemma for finite index languages generated by matrix grammars (see Lemma 3.1.6 in [21]). As this pumping lemma will be used to prove the proper inclusion of language families later in Chapter 7, it is recalled, for finite index languages generated by programmed grammars[7], by the following lemma.

Lemma 3.1.4. *Let* $n \geq 1$. *For every infinite language* $\mathrm{L} \in {}_{n}\mathbf{PRG}$, *there exists a word* $z \in \mathrm{L}$ *which can be written in the form*

$$z = u_1 v_1 w_1 x_1 u_2 v_2 w_2 x_2 \dots u_k v_k w_k x_k u_{k+1}$$

with $k \leq n$, $|v_1 x_1 v_2 x_2 \dots v_k x_k| > 0$, *and*

$$u_1 v_1^i w_1 x_1^i u_2 v_2^i w_2 x_2^i \dots u_k v_k^i w_k x_k^i u_{k+1} \in \mathrm{L}$$

for all $i \geq 1$.

Proof. See [21]. □

Regulated Automata

Regarding regulated grammars, it becomes natural to ask whether there exist automata counterparts accepting the same language families as well. In [43], Kolář and Meduna equipped pushdown automata with the same regulating device that was used in regular-controlled grammars—a *control language*. Like in regular-controlled grammars, a word is accepted by a *regulated pushdown automaton* if and only if it is accepted by the underlying pushdown automaton and simultaneously its parse belongs to the control language.

The family of languages accepted by regulated pushdown automata with a regular control language coincides with the family of context-free languages. This is a difference in comparison to regular-controlled grammars, where the language family strictly lies between context-free and context-sensitive families of languages. With a linear control language, regulated pushdown automata are able to accept every recursively enumerable language (see [43] for proofs).

In [44], Kolář and Meduna show that every recursively enumerable language can be also accepted by *one-turn atomic regulated pushdown automata* controlled by *linear* control language.

[7] Recall that ${}_{n}\mathbf{PRG}$ coincides with the family of languages of index n generated by matrix grammars.

An interesting way of regulation based on memorizing applied rules is used in *self-regulating finite automata*, introduced in [62] by Masopust and Meduna, where regulation is carried out in the following way: a self-regulating finite automaton M reads one input symbol, makes a move, and records used rule until it enters the special state, called the *turn state*. After that point, called a *turn*, M selects a rule associated with the first recorded rule and continues doing moves. If M works in *all-move* mode, the next rule to be selected is the rule associated with the second recorded rule, etc. If M does not work in all-move mode, it works in the *first-move* mode. Note that during the processing of recorded rules, M records the sequence of following rules as well. After the last recorded rule has been processed, M makes a turn and starts processing the new sequence of recorded rules. The process continues until M reads its whole input. According to the number of turns and the move mode, self-regulating finite automata are divided into *n-turn first-move self-regulating finite automata* and *n-turn all-move self-regulating finite automata*.

Observe that if M makes m moves from its start state to its turn state before the first turn, the number of moves that M makes to accept its input must be divisible by m.

Meduna and Masopust also showed in [61, 62] that for some $n \geq 0$, n-turn first-move self-regulating finite automata accept the family of languages generated by $(n+1)$-parallel right linear grammars and n-turn all-move self-regulating finite automata accept the family of languages generated by $(n+1)$-right linear simple matrix grammars.

There exist also self-regulating pushdown automata (see [61]), which are capable of accepting every recursively enumerable language if they are working in all-move mode.

Both regulated pushdown automata and self-regulating finite automata base their regulation on a prescribed sequence of rules and on rules that were used previously, respectively. In deep pushdown automata, introduced by Meduna in [68] (and by Křivka and Meduna in [50] under the name *deep top-down parsers*), moves are regulated by the pushdown content. More precisely, every rule in a deep pushdown automaton M has a number associated with it, called *depth*. M works like an ordinary pushdown automaton except that during the application of a rule of depth m it rewrites the mth topmost pushdown symbol with a word.

Definition 3.1.5. A *deep pushdown automaton* (abbreviated DPDA), M, is a septuple $\mathrm{M} = (\mathrm{Q}, \Sigma, \Gamma, \mathrm{R}, s, \mathrm{S}, \mathrm{F})$, where

- Q, Σ, Γ, s, S, and F are defined as in pushdown automaton, Q, Γ, and $\mathbb{N}$ are pairwise disjoint, $\Sigma \subseteq \Gamma$, and $\# \in (\Gamma - \Sigma)$ is the special symbol, called *bottom symbol*;
- $\mathrm{R} \subseteq (\mathbb{N}\mathrm{Q}(\Gamma - (\Sigma \cup \{\#\})) \times \mathrm{Q}(\Gamma - \{\#\})^{+}) \cup (\mathbb{N}\mathrm{Q}\{\#\} \times \mathrm{Q}(\Gamma - \{\#\})^{*}\{\#\})$ is a finite set of *rules*.

A rule $(mp\mathrm{A}, qx) \in \mathrm{R}$ is usually written as $mp\mathrm{A} \to qx$ and it is said to be a *rule of depth m*. If there is a rule $np\mathrm{A} \to qx \in \mathrm{R}$ such that for every rule $mp'\mathrm{A}' \to q'x' \in \mathrm{R}$ it holds $n \geq m$, then M is said to be of *depth n*. The depth of M will be denoted as depth(M).

Set $\Xi = \mathrm{Q} \times \Sigma^{*} \times (\Gamma - \{\#\})^{*}\{\#\}$. Then, $\chi \in \Xi$ is said to be a *configuration* of M.

The M-based relation of *direct pop move*, ${}_{p}\vdash_{\mathrm{M}}$, over Ξ is defined as follows:

$$(p, aw, ay) \ {}_{p}\vdash_{\mathrm{M}} (p, w, y)$$

where $p \in Q$, $a \in \Sigma$, $w \in \Sigma^*$, and $y \in \Gamma^*$.

The M-based relation of *direct expansion move*, ${}_{e}\vdash_{M}$, over Ξ is defined as follows:

$$(p, w, uAv)\ {}_{e}\vdash_{M}\ (q, w, uxv)$$

if and only if $mpA \rightarrow qx \in R$ and $\mathscr{O}_{\Gamma-\Sigma}(u) = m - 1$, where $p, q \in Q$, $w \in \Sigma^*$, $A \in \Gamma$, and $u, v, x \in \Gamma^*$. A move

$$(p, w, uAv)\ {}_{e}\vdash_{M}\ (q, w, uxv)\, [mpA \rightarrow qx]$$

is also referred as an *expansion of depth m*.

The M-based relation of *direct move*, $\vdash_M$, is then defined as $\vdash_M = {}_{p}\vdash_M \cup {}_{e}\vdash_M$. For $k \geq 0$, ${}_{p}\vdash^k_M$, ${}_{e}\vdash^k_M$, $\vdash^k_M$, ${}_{p}\vdash^+_M$, ${}_{e}\vdash^+_M$, $\vdash^+_M$, ${}_{p}\vdash^*_M$, ${}_{e}\vdash^*_M$, and $\vdash^*_M$ are defined as usual.

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

$$L(M) = \{w \mid (s, w, S\#) \vdash^*_M (f, \varepsilon, \#), w \in \Sigma^*, f \in F\}$$

The language accepted by M by empty pushdown, $L(M)_\varepsilon$, is defined as

$$L(M)_\varepsilon = \{w \mid (s, w, S\#) \vdash^*_M (q, \varepsilon, \#), w \in \Sigma^*, q \in Q\}$$

Let $\mathscr{M}$(DPDA) denote the set of all deep pushdown automata. Then, for every $k \geq 1$,

$$\begin{aligned} \mathbf{DPDA}_k &= \{L(M) \mid M \in \mathscr{M}(\mathrm{DPDA}), 1 \leq \mathrm{depth}(M) \leq k\} \\ \mathbf{DPDA}^\varepsilon_k &= \{L(M)_\varepsilon \mid M \in \mathscr{M}(\mathrm{DPDA}), 1 \leq \mathrm{depth}(M) \leq k\} \end{aligned}$$

denote the language families accepted by deep pushdown automata. □

Accepting power of deep pushdown automata of depth m coincide with the generative capacity of m-limited state grammars (discussed in the next subsection). Thus, language families accepted by deep pushdown automata form the infinite hierarchy of language families between context-free and context-sensitive language families.

Theorem 3.1.6. *Let* $n \geq 1$. *Then,* $\mathbf{DPDA}_n = \mathbf{DPDA}^\varepsilon_n = \mathbf{ST}^{-\varepsilon}_n$.

Proof. See [50, 68]. □

Another example of regulated automata are *k-counter automata* (see [31]). A k-counter automaton is a finite automaton equipped with k integer counters as additional storage. Two counters are enough to accept every recursively enumerable language.

Combination of Grammar and Automata Approach

Up till now, only regulated automata and grammars were discussed. However, there are also rewriting systems that combine parts that are essential to either grammars or automata. Such a rewriting system, for example, are *state grammars* (see [36]), where derivation steps are driven by states. Specifically, a rule r in a state grammar G is consisting of two parts—a context-free part and a transition part. r can be applied if and only if its state is the same as the current state of G and simultaneously a nonterminal symbol A to be rewritten is contained in the current G's sentential form. If these conditions are met, then the leftmost occurrence of A is rewritten and simultaneously G changes its current state according to r.

Definition 3.1.7. A *state grammar* (abbreviated STG), G, is a sextuple $\mathrm{G} = (\mathrm{V}, \mathrm{T}, \mathrm{K}, \mathrm{P}, \mathrm{S}, s)$, where

- V is a *total alphabet*;
- $\mathrm{T} \subseteq \mathrm{V}$ is an alphabet of *terminal symbols*;
- K is a finite set of *states*, $\mathrm{K} \cap \mathrm{V} = \emptyset$;
- $\mathrm{P} \subseteq (\mathrm{V} - \mathrm{T}) \times \mathrm{K} \times \mathrm{V}^* \times \mathrm{K}$ is a finite set of *rules*;
- $\mathrm{S} \in (\mathrm{V} - \mathrm{T})$ is the *start symbol*;
- $s \in \mathrm{K}$ is the *initial state*.

A rule $(\mathrm{A}, p, x, q) \in \mathrm{P}$ is usually written as $(\mathrm{A}, p) \rightarrow (x, q)$. G is said to be *propagating* if and only if $(\mathrm{A}, p) \rightarrow (x, q) \in \mathrm{P}$ implies $x \neq \varepsilon$.

The G-based relation of *direct derivation*, $\Rightarrow_{\mathrm{G}}$, over $\mathrm{V}^* \times \mathrm{K}$ is defined as follows: $(u\mathrm{A}v, p) \Rightarrow_{\mathrm{G}} (uxv, q)$ if and only if $(\mathrm{A}, p) \rightarrow (x, q) \in \mathrm{P}$ and for every $(\mathrm{B}, p) \rightarrow (y, t) \in \mathrm{P}$, $\mathrm{B} \notin \mathrm{alph}(u)$, where $p, q, t \in \mathrm{K}$, $\mathrm{A}, \mathrm{B} \in (\mathrm{V} - \mathrm{T})$, and $u, v, x, y \in \mathrm{V}^*$.

Let $k \geq 1$. The G-based relation of k*-limited direct derivation*, ${}_k\Rightarrow_{\mathrm{G}}$, over $\mathrm{V}^* \times \mathrm{K}$ is defined as follows: $(u\mathrm{A}v, p)\ {}_k\Rightarrow_{\mathrm{G}} (uxv, q)$ if and only if $(\mathrm{A}, p) \rightarrow (x, q) \in \mathrm{P}$, $\mathscr{O}_{\mathrm{V}-\mathrm{T}}(u\mathrm{A}) \leq k$, and for every $(\mathrm{B}, p) \rightarrow (y, t) \in \mathrm{P}$, $\mathrm{B} \notin \mathrm{alph}(u)$, where $p, q, t \in \mathrm{K}$, $\mathrm{A}, \mathrm{B} \in (\mathrm{V} - \mathrm{T})$, and $u, v, x, y \in \mathrm{V}^*$.

For $l \geq 0$, $\Rightarrow^l_{\mathrm{G}}$, ${}_k\Rightarrow^l_{\mathrm{G}}$, $\Rightarrow^*_{\mathrm{G}}$, ${}_k\Rightarrow^*_{\mathrm{G}}$, $\Rightarrow^+_{\mathrm{G}}$, and ${}_k\Rightarrow^+_{\mathrm{G}}$ are defined as usual.

The language generated by G, L(G), is defined as

$$\mathrm{L}(\mathrm{G}) = \{w \mid (\mathrm{S}, s) \Rightarrow^*_{\mathrm{G}} (w, q), w \in \mathrm{T}^*, q \in \mathrm{K}\}$$

The language generated by G in k-limited way, L(G, k), is defined as

$$\mathrm{L}(\mathrm{G}, k) = \{w \mid (\mathrm{S}, s)\ {}_k\Rightarrow^*_{\mathrm{G}} (w, q), w \in \mathrm{T}^*, q \in \mathrm{K}\}$$

$\mathbf{ST}$, $\mathbf{ST}^{-\varepsilon}$, $\mathbf{ST}_k$, and $\mathbf{ST}_k^{-\varepsilon}$ denote language families generated by state grammars, propagating state grammars, state grammars in k-limited way, and propagating state grammars in k-limited way, respectively. Set

$$\mathbf{ST}_\infty = \bigcup_{n \geq 1} \mathbf{ST}_n \qquad \mathbf{ST}_\infty^{-\varepsilon} = \bigcup_{n \geq 1} \mathbf{ST}_n^{-\varepsilon}$$

□

The following theorem summarizes generative capacity of state grammars. The identity **ST** = **RE** has been proven in [32], the rest has been proven in [36].

Theorem 3.1.8. *Let* $n \geq 1$. *Then,*

(a) $\mathbf{CF} = \mathbf{ST}_1^{-\varepsilon}$, $\mathbf{ST}_n^{-\varepsilon} \subset \mathbf{ST}_{n+1}^{-\varepsilon}$, $\mathbf{ST}_n^{-\varepsilon} \subset \mathbf{ST}_\infty^{-\varepsilon} \subset \mathbf{ST}^{-\varepsilon} = \mathbf{CS}$*;*

(b) $\mathbf{ST} = \mathbf{RE}$.

Proof. See [32, 36]. □

Observe that in [21], state grammars are defined with no requirement of rewriting the leftmost nonterminal symbol during the direct derivation step. As a consequence, state grammars from [21] have the same generative capacity as matrix grammars.

Finite state control as a regulating device is used also in *#-rewriting systems of finite index*, introduced by Křivka, Meduna, and Schönecker in [52]. #-rewriting systems work like state grammars except for these differences: there is no requirement of rewriting the leftmost nonterminal symbol, there is just one nonterminal symbol #, called *bounder*, and for some #-rewriting system Ω of index n, every configuration of Ω must contain no more than n occurrences of #.

Definition 3.1.9. A *#-rewriting system* (abbreviated #RS), Ω, is a quadruple $\Omega = (\mathrm{Q}, \Sigma, s, \mathrm{R})$, where

- Q is a finite set of *states*;
- Σ is an alphabet, $\# \in \Sigma$ is a special symbol called *bounder*, $\mathrm{Q} \cap \Sigma = \emptyset$;
- $s \in \mathrm{Q}$ is the *initial state*;
- $\mathrm{R} \subseteq \mathrm{Q} \times \mathbb{N} \times \{\#\} \times \mathrm{Q} \times \Sigma^*$ is a finite set of *rules*.

A rule $(p, n, \#, q, x) \in \mathrm{R}$ is usually written as $p_n\# \rightarrow qx$. A *configuration* of Ω is a word from $\mathrm{Q}\Sigma^*$.

The Ω-based relation of *direct rewriting step*, $\Rightarrow_\Omega$, over $\mathrm{Q}\Sigma^*$ is defined as follows: $pu\#v \Rightarrow_\Omega quxv$ if and only if $p_n\# \rightarrow qx \in \mathrm{R}$ and $\mathcal{O}_\#(u) = n - 1$, where $p, q \in \mathrm{Q}$, $n \in \mathbb{N}$, and $u, x, v \in \Sigma^*$. For $m \geq 0$, $\Rightarrow_\Omega^m$, $\Rightarrow_\Omega^*$, and $\Rightarrow_\Omega^+$ are defined as usual.

The language generated by Ω, L(Ω), is defined as

$$\mathrm{L}(\Omega) = \{w \mid s\# \Rightarrow_\Omega^* qw, q \in \mathrm{Q}, w \in (\Sigma - \{\#\})^*\}$$

Let $k \geq 1$. If $s\# \Rightarrow_{\Omega}^{*} qy$ implies $\mathscr{O}_{\#}(y) \leq k$, $q \in \mathrm{Q}$, $y \in \Sigma^{*}$, then Ω is said to be #-rewriting system of *index* k. The family of languages generated by #-rewriting systems of index k is denoted by $\mathscr{L}_{k}(\#\mathrm{RS})$. □

#-rewriting systems of index k have the same generative capacity as programmed grammars of index k.

Theorem 3.1.10. *Let* $k \geq 1$. *Then,* $\mathscr{L}_{k}(\#\mathrm{RS}) = {}_{k}\mathbf{PRG}$.

Proof. See [47, 52]. □

Jumping Rewriting

Regulating devices discussed so far aim to decrease the number of possible ways of rewriting a configuration of some rewriting system. For instance, in regular-controlled grammars, those applications of context-free rules that will lead to a parse not belonging to control language must be discarded even though such a parse can be valid in sense of context-free grammars.

Regulation based on jumping works in the opposite way. Instead of reducing the number of possible ways of rewriting, they are increased. To demonstrate it on example, take a definition of *jumping grammars*, introduced by Křivka and Meduna in [51]. In general, a jumping grammar is an unrestricted grammar with additional relations of direct derivation defined (see Definition 3.1.11). Given an unrestricted grammar $\mathrm{G} = (\mathrm{V}, \mathrm{T}, \mathrm{P}, \mathrm{S})$, four words $u, v, u', v' \in \mathrm{V}^{*}$, and a rule $x \rightarrow y \in \mathrm{P}$. Normally, $uxv \Rightarrow_{\mathrm{G}} u'yv'$ such that $u = u'$ and $v = v'$. However, if jumping is in effect, then $uxv \; {}_{j}\!\Rightarrow_{\mathrm{G}} u'yv'$ such that $uv = u'v'$. In other words, x is first erased from uxv and then y is inserted at any position inside uv. Thus, there are many more possibilities how to rewrite uxv according to $x \rightarrow y$ in jumping mode compared to the normal case.

Definition 3.1.11. Let $\mathrm{G} = (\mathrm{V}, \mathrm{T}, \mathrm{P}, \mathrm{S})$ be an unrestricted grammar. Let $u, v \in \mathrm{V}^{*}$. Define four G-based relations of direct derivation over V^{*} as follows:

(i) $u \; {}_{s}\!\Rightarrow_{\mathrm{G}} v$ if and only if $u \Rightarrow_{\mathrm{G}} v$;

(ii) $u \; {}_{lj}\!\Rightarrow_{\mathrm{G}} v$ if and only if $u = wtxz$, $v = wytz$, $x \rightarrow y \in \mathrm{P}$, and $w, t, z \in \mathrm{V}^{*}$;

(iii) $u \; {}_{rj}\!\Rightarrow_{\mathrm{G}} v$ if and only if $u = wxtz$, $v = wtyz$, $x \rightarrow y \in \mathrm{P}$, and $w, t, z \in \mathrm{V}^{*}$;

(iv) ${}_{j}\!\Rightarrow_{\mathrm{G}} = {}_{lj}\!\Rightarrow_{\mathrm{G}} \cup {}_{rj}\!\Rightarrow_{\mathrm{G}}$.

Let $h \in \{s, lj, rj, j\}$. For $k \geq 0$, ${}_{h}\!\Rightarrow_{\mathrm{G}}^{k}$, ${}_{h}\!\Rightarrow_{\mathrm{G}}^{*}$, and ${}_{h}\!\Rightarrow_{\mathrm{G}}^{+}$ are defined as usual.

The *language* generated by G in h-mode, $\mathrm{L}(\mathrm{G}, {}_{h}\!\Rightarrow)$, is defined as

$$\mathrm{L}(\mathrm{G}, {}_{h}\!\Rightarrow) = \{w \mid \mathrm{S} \; {}_{h}\!\Rightarrow_{\mathrm{G}}^{*} w, w \in \mathrm{T}^{*}\}$$

Let X denote the type of a grammar. Then,

$$\mathscr{L}(\mathrm{X}, {}_h\Rightarrow) = \{\mathrm{L}(\mathrm{G}, {}_h\Rightarrow) \mid \mathrm{G} \in \mathscr{G}(\mathrm{X})\}$$

denotes the family of languages generated by a grammar of type X in h-mode. □

To show how jumping rewriting affects the generative capacity of unrestricted grammars, recall some results from [51]. Normally, regular grammars generate the family of languages that is strictly included in the family of context-free languages. In jumping mode, regular grammars are able to generate languages that are not context-free, but there also exist regular languages that cannot be generated by any regular grammar working in jumping mode. That is $\mathscr{L}(\mathrm{RG}, {}_j\Rightarrow) - \mathbf{CF} \neq \emptyset$ and $\mathbf{REG} - \mathscr{L}(\mathrm{RG}, {}_j\Rightarrow) \neq \emptyset$.

In the case of unrestricted grammars, jumping has no influence on their generative capacity. This is because rules of unrestricted grammars work with context and thus if a right-hand side of a rule is inserted in the wrong place during a derivation step, the entire derivation is blocked. Therefore, $\mathscr{L}(\mathrm{UG}, {}_j\Rightarrow) = \mathbf{RE}$.

Besides jumping grammars, there also exist jumping automata. In [71], Meduna and Zemek introduced *jumping finite automata*. Just like in the case of jumping grammars, jumping finite automata differ from ordinary finite automata in the definition of how a direct move is performed. If a jumping automaton M is in a state p and it is going to perform a move according to some rule $pa \rightarrow q$, then (1) M moves the reading head to an arbitrary position on its input tape such that the next symbol to be read will be a, (2) M reads a by erasing it from the input tape, and (3) M changes its state to q. Simply said, the reading head literally "jumps" over M's input tape. Meduna and Zemek also introduced *general jumping finite automata* that work exactly like jumping finite automata but they are capable of reading words instead of symbols from their input. Meduna and Křivka (see [51]) show that $\mathscr{L}(\mathrm{RG}, {}_j\Rightarrow)$ and $\mathscr{L}(\mathrm{RLG}, {}_j\Rightarrow)$ coincide with the families of languages accepted by jumping finite automata and general jumping finite automata, respectively. Properties and complexity of jumping finite automata were also studied in [22, 23].

Other examples of jumping grammars and automata that can be found in the literature are *one-way jumping finite automata* (see [8]), *double-jumping finite automata* (see [40]), *jumping 5'→3' Watson-Crick finite automata* (see [41]), or *jumping scattered context grammars* (see [69]).

3.2 Parallel and Cooperative Rewriting

The previous section discussed rewriting systems in which only one rewriting rule is applied during a single rewriting step. This section focuses on rewriting systems where during a single rewriting step more than one rule can be applied simultaneously. Additionally, this section also focuses on multi-component rewriting systems where components are working under some level of cooperation.

Parallel Rewriting

A well-known example of rewriting systems with parallel rewriting are *Lindenmayer systems* (or briefly *L-systems*), named according to their inventor, biologist Aristid Lindenmayer (see [56, 57]). In the original L-systems, later called 0L systems, all symbols are terminal symbols and during a single rewriting step every symbol must be rewritten. Rewriting rules in 0L systems are context-free with a terminal alphabet as their domain.

Definition 3.2.1. An *0L system*, H, is a triple $\mathrm{H} = (\Sigma, \mathrm{P}, \sigma)$, where Σ is an alphabet of terminal symbols, $\mathrm{P} \subseteq \Sigma \times \Sigma^*$ is a finite set of rules, $\mathrm{domain(P)} = \Sigma$, and $\sigma \in \Sigma^+$ is the axiom. H is said to be *propagating* if $a \to x \in \mathrm{P}$ implies $x \neq \varepsilon$.

The H-based relation of direct derivation, $\Rightarrow_{\mathrm{H}}$, over Σ^* is defined as follows: $u \Rightarrow_{\mathrm{H}} v$ if and only if $u = a_1 a_2 \ldots a_n$, $v = x_1 x_2 \ldots x_n$, and $a_i \to x_i \in \mathrm{P}$, where $a_i \in \Sigma$, $x_i \in \Sigma^*$, $1 \leq i \leq n$, and $n \geq 1$. For $k \geq 0$, $\Rightarrow_{\mathrm{H}}^k$, $\Rightarrow_{\mathrm{H}}^+$, and $\Rightarrow_{\mathrm{H}}^*$ are defined as usual.

The language generated by H, L(H), is defined as $\mathrm{L(H)} = \{w \mid \sigma \Rightarrow_{\mathrm{H}}^* w, w \in \Sigma^*\}$. **0L** and $\mathbf{0L}^{-\varepsilon}$ denote families of languages generated by 0L systems and propagating 0L systems, respectively. □

In [85], Rozenberg and Doucet show that $\mathbf{0L} \subset \mathbf{CS}$ and also that **0L** is incomparable with **CF** and **REG**. Chapter 5 in [88] gives a survey of other variants of L-systems that were investigated, like E0L systems, which are 0L systems extended about nonterminal symbols so they are able to generate every context-free language.

The following lemma from [85] will be used in further proofs.

Lemma 3.2.2. *Let* G *be a 0L system. Then there exists a number k such that for every word* $w \in \mathrm{L(G)}$ *there exists a derivation such that* $|u| \leq k|w|$ *for every word u in that derivation.*

L-systems are an example of rewriting systems that work in a fully parallel way. The *n-parallel right linear grammars* (see [79]) and *n-right linear simple matrix grammars* (see [33]) are fully parallel in the sense of rewriting nonterminal symbols. In an n-parallel right linear grammar, the start symbol is first rewritten to a sentential form containing n nonterminal symbols and then during every derivation step all nonterminal symbols are simultaneously rewritten by application of n right linear rules. A derivation step in n-right linear simple matrix grammars is performed similarly but in addition rules that are applied during a single derivation step must be organized into a tuple, called matrix, where the first rule from the matrix is applied on the first leftmost nonterminal symbol, the second rule to the second one, and so on. Both n-parallel right linear grammars and n-right linear simple matrix grammars are capable of generating languages that are not context-free, they induce an infinite hierarchy of language families, and their language families, are strictly included in **CS** (see [80, 97]).

Parallel rewriting can be also observed in *scattered context grammars* (see [28]). The way how scattered context grammars work is very similar to the one seen in n-right linear simple matrix grammars. The rules are also organized in tuples, but they are context-free and not all nonterminal symbols in a sentential form must be rewritten during a single derivation step (however, all of the rules from a tuple must be applied). Observe that organizing rules into tuples provides a tool for transferring information from one part of a sentential form to another. This, together with context-free rules and partial parallelism[8], gives scattered context grammars the ability to generate any recursively enumerable language (see [64, 65]).

More examples of grammars with parallel rewriting can be found in the literature (see [37–39, 46, 55]).

In automata, parallel rewriting is usually performed via multiple reading and/or writing heads that operate over an input tape (or tapes). Some examples of such a kind of automata are *multi-head finite automata* (see [81]) or *n-parallel jumping finite automata* (see [42]). An n-parallel jumping finite automaton M has its input tape divided into n regions, where each region has its own reading head and a state attached to it. Each head processes its region independently of other heads by performing jumping moves over the region, like in jumping finite automata. A word is accepted by M if all symbols from every region have been read and every region has the state associated with it considered as final. In [42], Meduna and Kocman studied several modes of jumping. For a mode where jumping is performed in right direction on a tape, n-parallel jumping finite automata accept exactly the same family of languages that is generated by n-parallel right linear grammars.

Multi-Component Rewriting Systems

Rewriting systems discussed so far were treated as independent language modeling devices. However, rewriting systems can be organized into larger structures, which become also rewriting systems. Rewriting systems that are parts of such structures are commonly called *components* and these structures can thus be generally referred to as multi-component rewriting systems. Grammar and automata systems are examples of multi-component rewriting systems where components are grammars and automata, respectively.

T0L and *ET0L systems* can also be considered to be multi-component rewriting systems (see [83, 84]). Like in 0L systems, a T0L system consists of an alphabet, the axiom, but contrary to 0L system it can contain more than one set of rules, called *tables* (hence the letter "T" in the name). In other words, a T0L system consists of the finite number of 0L systems as its components with a shared alphabet and axiom. During a single rewriting step, only one component can perform a rewriting. ET0L systems work exactly like T0L systems but they can additionally use nonterminal symbols.

A similar approach to rewriting as in T0L and ET0L systems can be observed also in *cooperating/distributed grammar systems* (see [14]). A cooperating/distributed grammar system Γ of degree n consists of n context-free grammars as its components, where components have a common alphabet of terminal and nonterminal symbols and the start symbol. The sentential form is also shared between components. When Γ starts its derivation process, the ith component of Γ, G_i, is selected to perform a sequence of derivation steps on the shared sentential form. After a certain number of derivation steps, G_i stops so another component of Γ can take its place. The whole process is repeated until the shared sentential form becomes a word containing only terminal symbols.

[8] With full parallelism, scattered context grammars will be of the finite index and then of less generative capacity than context-sensitive grammars (see [21]).

A number of derivation steps performed by a single component can be further restricted by modes of derivation. A component of a cooperating/distributed grammar system can work under four modes of derivation, namely *terminating derivation* (component performs as many derivation steps as possible), *k-steps derivation* (component performs exactly k derivation steps), *at least k-steps derivation* (component performs at least k derivation steps), and *at most k-steps derivation* (component performs at most k-derivation steps). With no mode, a component can perform as many derivation steps as desirable. In ordinary cooperating/distributed grammar systems, all components must be working under the same derivation mode. This does not hold for *hybrid cooperating/distributed grammar systems* (see [74, 77]), where the derivation mode of specific components can be different. Cooperating/distributed grammar systems generate families of languages that lie between context-free and context-sensitive language families. Cooperating/distributed grammar systems with three components working in terminating derivation mode generate the language family generated by ET0L systems (see [15]).

T0L, ET0L, and cooperating/distributed grammar systems are examples of multi-component rewriting systems where only one component can work on a sentential form in a given time and the others must wait. Components in *parallel communicating grammar systems* and *multi-generative grammar systems*, on the other hand, work simultaneously.

Like in cooperating/distributed grammar systems, a parallel communicating grammar system Γ of degree n (see [91]) also consists of n context-free grammars as its components with shared alphabets of the terminal and nonterminal symbols. What is different is that each component works on its own sentential form and some nonterminal symbols are treated as *query symbols*. The derivation process in Γ is performed in the following way. In the beginning, all components work independently of each other on derivation steps over their sentential forms. When some component, G_i, produces a query symbol, which works as a reference to another component, G_j, the derivation process is stopped, the query symbol is replaced by G_j's sentential form, and then the derivation process continues again. The language of Γ is the language of Γ's first component.

Parallel communicating grammar systems can work in *returning mode* (the requested component, that is a component whose sentential form was used to replace a query symbol, starts a next derivation step from its start symbol) or in *non-returning mode* (the requested component starts the next derivation from its current sentential form). They can be also *centralized*, which means that only the first component is allowed to produce query symbols.

Parallel communicating grammar systems with components containing only regular rules induce an infinite hierarchy of language families with respect to the number of components (see [35, 78]). The language family generated by matrix grammars is strictly included in the language family generated by parallel communicating grammar systems with context-free components (see [73]). However, parallel communicating grammar systems with context-free components are still less powerful than unrestricted grammars (see [15]).

Multi-generative grammar systems were introduced by Lukáš and Meduna in [58, 59]. A multi-generative grammar system Γ consists of n context-free grammars and a finite control set. Γ is said to be *canonical* if all its components are restricted to do only leftmost[9] derivations, *general* if there are no restrictions placed on its components, or *hybrid* if at least one component, but not all, are restricted to do only leftmost derivations.

[9] In leftmost derivations, only the leftmost nonterminal symbol of a sentential form can be rewritten.

Each component of Γ works on its sentential form in a synchronized way. That is, Γ can perform a single derivation step if and only if all of its components can perform a single derivation step and simultaneously, either rules used must form a tuple from the Γ's control set if Γ is *rule synchronized* or left-hand sides of rules used must form a tuple from the Γ's control set if Γ is *nonterminal synchronized*.

There are three types of languages generated by Γ: the language generated by the first component of Γ, the language consisting of all words generated by all components of Γ, and the language that contains the words formed by the concatenation of words of particular components of Γ.

Whether multi-generative grammar systems are rule synchronized or nonterminal synchronized has no impact on their generative capacity. Canonical multi-generative grammar systems with two components have enough power to generate every recursively enumerable language (see [58]).

Concerning grammar systems, it became natural to study their automata counterparts as well. *Parallel communicating finite automata systems*, introduced by Mitrana, Mateescu, and Martín-Vide in [60], are based on a similar principle as parallel communicating grammar systems. In a parallel communicating finite automata system M, every finite automaton component has its own state and input tape and thus it works independently on others. Communication between components is done via *query states*. When some component moves to query state, its next state is the state of the requested component and, if M works in returning mode, the next state of the requested component becomes its initial state. At the start, all components of M have the same content written on their input tapes.

In *parallel communicating pushdown automata systems* (see [17]), the communication between components is based on *query pushdown symbols*. When a pushdown query symbol become the topmost one in a component's pushdown, it is replaced by the content of the pushdown of the requested component. In returning mode, the next step is resetting the requested component's pushdown to its initial content.

Parallel communicating automata systems are more powerful than their grammatical counterparts. For instance, parallel communicating pushdown automata systems working in returning mode and with three components are able to accept every recursively enumerable language (see [17]).

As an answer to multi-generative grammar systems, Čermák and Meduna introduce two variants of *n-accepting restricted pushdown automata systems* (see [6, 7]). In both variants, components work simultaneously and are synchronized by a finite control set. The difference lies in how components are synchronized.

In the first variant, *n-accepting state-restricted pushdown automata systems*, each component has assigned status that indicates whether the component is enabled or disabled and a control set containing so-called *switch rules* that specify a relation between components states and their statuses. When a system is going to perform a move, the first step is that all enabled components perform their moves and all disabled components are stay in their current configuration. Then, according to the states of all components, the corresponding switch rule is selected and according to this rule statuses of all components are changed. If such a switch rule does not exist, the statuses of all components remain unchanged.

In the second variant, *n-accepting move-restricted pushdown automata systems*, components are synchronized exactly in the same way as in rule synchronized multi-generative grammar systems.

Components in both variants of *n*-accepting restricted pushdown automata systems accept their input by empty pushdown. Furthermore, both variants of *n*-accepting restricted pushdown automata systems have the same power and they are language equivalents to canonical multi-generative grammar systems of the same degree (see [6]).

The list of so far discussed grammar and automata systems is not comprehensive, but it demonstrates that when rewriting systems cooperate with each other their language descriptive power may increase. More examples of grammar and automata systems can be found in the literature (see [4, 16, 18, 34]).

Part II
New Language Models

This part forms the core of this thesis. It presents four new language models, published in papers *On state-synchronized automata systems* ([53]), *Absolutely unlimited deep pushdown automata* ([54]), *Jumping pure grammars* ([49]), and *On k#\$-rewriting systems* ([48]). It also presents extra material not included in mentioned papers due to the size limit. The content of this part is divided into four chapters.

Chapter 4 presents *state-synchronized automata systems*. It gives formal definitions of such automata systems and demonstrates them on examples. Then, it discusses their accepting power concerning the number and type of components.

Chapter 5 presents two kinds of unlimited deep pushdown automata—*absolutely unlimited deep pushdown automata* (published in [54]) and *relatively unlimited deep pushdown automata* (yet unpublished). It gives their formal definitions, demonstrates them on examples, and investigates their accepting power.

Chapter 6 presents *jumping pure grammars*. First, it defines jumping pure grammars and their language families via modes of direct derivations. Then, it investigates the mutual relations of language families generated by jumping pure grammars with context-free rules.

Chapter 7 presents *k#\$-rewriting systems*. It defines them formally, shows them on example, and compares their generative capacity with programmed grammars of index k and state grammars working in k-limited way. This chapter also gives the full formal proofs of their shortened versions published in [48].

Chapter 4
State-Synchronized Automata Systems

This chapter presents state-synchronized automata systems that were introduced in [53]. State-synchronized automata systems were inspired by the previous work on automata systems reviewed in Chapter 3. The aim was to find a new type of automata system whose components work simultaneously and the communication between components is secured by states, but in a simpler way than in parallel communicating finite automata systems or n-accepting state-restricted pushdown automata systems. As a consequence, components in state-synchronized automata systems are controlled by words from a finite control language over states of particular components and a move is performed if and only if all components can do their move simultaneously and their states form a word from control language.

The content of this chapter is divided into two sections. Section 4.1 gives a formal definition of state-synchronized automata systems and demonstrates them on examples. Section 4.2 then studies state-synchronized automata systems with finite, pushdown, and one-turn pushdown automata as components and investigates their accepting power.

4.1 Definitions and Examples

Recall the definition of state-synchronized automata system published in [53].

Definition 4.1.1. Let n be a positive integer. A *state-synchronized automata system* of degree n (abbreviated SCAS$_n$) is an $(n+1)$-tuple $\Gamma = (M_1, M_2, \dots, M_n, \Psi)$, where M_i is an FA or a PDA, and it is referred to as the ith *component* of Γ, for all $1 \leq i \leq n$. $\Psi \subseteq Q_1 Q_2 \dots Q_n$ is a *control language* of Γ, where Q_i is the set of states in M_i, $1 \leq i \leq n$. Furthermore, Σ, Γ_i, s_i, S_i, and F_i denote the input alphabet of Γ, the pushdown alphabet of M_i, the initial state of M_i, the initial symbol on M_i's pushdown, and the set of final states in M_i, respectively, for all $1 \leq i \leq n$. If M_i is an FA, then set $\Gamma_i = \emptyset$ and $S_i = \varepsilon$, for all $1 \leq i \leq n$.

A *configuration* of Γ is an n-tuple $(\chi_1, \chi_2, \dots, \chi_n)$, where χ_i is a configuration of M_i, for all $1 \leq i \leq n$.

Let π_i be the mapping from $\Gamma_i^* Q_i \Sigma^*$ to Q_i such that $\pi_i(x_i q_i w) = q_i$, $x_i \in \Gamma_i^*$, $q_i \in Q_i$, $w \in \Sigma^*$, for all $1 \leq i \leq n$. Furthermore, let $\alpha = (\chi_1, \chi_2, \dots, \chi_n)$ and $\alpha' = (\chi_1', \chi_2', \dots, \chi_n')$ be two configurations of Γ. The Γ-based relation of *direct move*, $\vdash_\Gamma$, is defined as follows: if for every $1 \leq i \leq n$ it holds that $\chi_i \vdash_{M_i} \chi_i'$, and $\pi_1(\chi_1)\pi_2(\chi_2)\ldots\pi_n(\chi_n) \in \Psi$, then $\alpha \vdash_\Gamma \alpha'$. For $k \geq 0$, $\vdash_\Gamma^k$, $\vdash_\Gamma^*$, and $\vdash_\Gamma^+$ are defined as usual.

Analogously to PDA, define three types of *languages accepted by* Γ as

$$\begin{aligned}
L(\Gamma)_f &= \left\{ w \;\middle|\; \begin{array}{l} (S_1s_1w, S_2s_2, \ldots, S_ns_n) \vdash_\Gamma^* (\gamma_1f_1, \gamma_2f_2, \ldots, \gamma_nf_n) \\ \gamma_i \in \Gamma_i^*, f_i \in F_i, 1 \leq i \leq n, w \in \Sigma^* \end{array} \right\} \\
L(\Gamma)_\varepsilon &= \left\{ w \;\middle|\; \begin{array}{l} (S_1s_1w, S_2s_2, \ldots, S_ns_n) \vdash_\Gamma^* (q_1, q_2, \ldots, q_n) \\ q_i \in Q_i, q_i \in F_i \text{ if } M_i \text{ is an FA}, 1 \leq i \leq n, w \in \Sigma^* \end{array} \right\} \\
L(\Gamma)_{f\varepsilon} &= \left\{ w \;\middle|\; \begin{array}{l} (S_1s_1w, S_2s_2, \ldots, S_ns_n) \vdash_\Gamma^* (f_1, f_2, \ldots, f_n) \\ f_i \in F_i, 1 \leq i \leq n, w \in \Sigma^* \end{array} \right\}
\end{aligned}$$

□

The following example demonstrates the capability of $SCAS_n$ to accept a language which is not context-free.

Example 4.1.2. Let $\Gamma = (M, M', \Psi)$ be an $SCAS_2$, where M, M' are PDAs defined as

- $M = (\{s, q_a, q_b, q_c, f\}, \{a, b, c\}, \{S, A, B, C\}, R, s, S, \{f\})$, where R contains rules

$$\begin{array}{llll} Ssa \rightarrow SAq_a & Aq_aa \rightarrow AAq_a & Aq_ab \rightarrow Aq_b & Aq_bb \rightarrow Aq_b \\ Aq_bc \rightarrow q_c & Aq_cc \rightarrow q_c & Sq_c \rightarrow f & \end{array}$$

- $M' = (\{s', q_a', q_b', q_c', f'\}, \{a, b, c\}, \{S, A, B, C\}, R', s', S, \{f'\})$, where R contains rules

$$\begin{array}{llll} Ss' \rightarrow Sq_a' & Sq_a' \rightarrow Sq_a' & Sq_a' \rightarrow SBq_b' & Bq_b' \rightarrow BBq_b' \\ Bq_b' \rightarrow q_c' & Bq_c' \rightarrow q_c' & Sq_c' \rightarrow f' & \end{array}$$

and Ψ is a control language of Γ defined as $\Psi = \{ss', q_aq_a', q_bq_b', q_cq_c'\}$. The word *aaabbbccc* is accepted by Γ in this way

$$\begin{array}{rll}
(Ssaaabbbccc, Ss') & \vdash_\Gamma & (SAq_aaabbbccc, Sq_a') \\
& \vdash_\Gamma & (SAAq_aabbbccc, Sq_a') \\
& \vdash_\Gamma & (SAAAq_abbbccc, Sq_a') \\
& \vdash_\Gamma & (SAAAq_bbbccc, SBq_b') \\
& \vdash_\Gamma & (SAAAq_bbccc, SBBq_b') \\
& \vdash_\Gamma & (SAAAq_bccc, SBBBq_b') \\
& \vdash_\Gamma & (SAAq_ccc, SBBq_c') \\
& \vdash_\Gamma & (SAq_cc, SBq_c') \\
& \vdash_\Gamma & (Sq_c, Sq_c') \\
& \vdash_\Gamma & (f, f')
\end{array}$$

Clearly, $L(\Gamma)_f = L(\Gamma)_\varepsilon = L(\Gamma)_{f\varepsilon} = \{a^nb^nc^n \mid n \geq 1\}$. □

Observe that $SCAS_n$ Γ, where $n \geq 1$, do not need to contain a word in Ψ which is formed from states from a final configuration to make a successful final computation step. On the other hand, as will be shown later in Definition 4.1.3, it would be useful to introduce Ψ_f as

$$\Psi_f = \Psi \cup \left\{ q_1 q_2 \dots q_n \,\middle|\, \begin{array}{l} (\gamma_1 q_1, \gamma_2 q_2, \dots, \gamma_n q_n) \text{ is a final configuration of } \Gamma \\ \gamma_i \in \Gamma_i^*, q_i \in Q_i, 1 \leq i \leq n \end{array} \right\}$$

Let Γ be an SCAS$_n$, for some $n \geq 1$. Define $\mathcal{R}_\Gamma$ as the Cartesian product $\mathcal{R}_\Gamma = R_1 \times R_2 \times \dots \times R_n$, where R_i is the set of rules of the ith component of Γ, for all $1 \leq i \leq n$. If $\alpha \in \mathcal{R}_\Gamma$, then $\alpha(i)$ denotes the ith element of α, and clearly $\alpha(i) \in R_i$, for all $1 \leq i \leq n$.

Define the mapping π_l from $\mathcal{R}_\Gamma$ to $Q_1 Q_2 \dots Q_n$ as

$$\pi_l(\alpha) = \pi_1(\text{lhs}(\alpha(1)))\pi_2(\text{lhs}(\alpha(2))) \dots \pi_n(\text{lhs}(\alpha(n)))$$

That is π_l maps an n-tuple of rules

$$(A_1 p_1 a_1 \rightarrow x_1 q_1, A_2 p_2 \rightarrow x_2 q_2, \dots, A_n p_n \rightarrow x_n q_n)$$

to the word $p_1 p_2 \dots p_n$. The mapping π_r from $\mathcal{R}_\Gamma$ to $Q_1 Q_2 \dots Q_n$ is defined analogously as $\pi_r(\alpha) = \pi_1(\text{rhs}(\alpha(1)))\pi_2(\text{rhs}(\alpha(2))) \dots \pi_n(\text{rhs}(\alpha(n)))$.

Now, let introduce the definition of determinism in SCAS$_n$.

Definition 4.1.3. Let Γ be an SCAS$_n$, for some $n \geq 1$. Then Γ is said to be *deterministic* (abbreviated dSCAS$_n$) if for every $\alpha \in \mathcal{R}_\Gamma$, such that $\pi_l(\alpha) \in \Psi$ and $\pi_r(\alpha) \in \Psi_f$, the following holds

$$\text{card}(\{\alpha' \mid \alpha' \in \mathcal{R}_\Gamma, \pi_r(\alpha') \in \Psi_f, \text{lhs}(\alpha'(i)) \in \text{subword}(\text{lhs}(\alpha(i))), 1 \leq i \leq n\}) = 1$$

□

Clearly, the SCAS$_2$ from Example 4.1.2 is deterministic. The following example shows the difference between deterministic and nondeterministic SCAS$_n$ according to Definition 4.1.3.

Example 4.1.4. Consider SCAS$_2$ $\Gamma = (M_1, M_2, \Psi)$, where M_1, M_2 are PDAs defined as

- $M_1 = (\{s, q_1, q_2, f\}, \{a, b\}, \{S, A, B\}, \{$

$$\begin{array}{ll} r_1\colon Ssa \rightarrow Sq_1, & r_3\colon Sq_1 \rightarrow Sf \\ r_2\colon Ssb \rightarrow Sq_2, & r_4\colon Sq_2 \rightarrow Sf \end{array}$$

  $\}, s, S, \{f\})$

- $M_2 = (\{s', q_1', q_2', f'\}, \{a, b\}, \{S, A, B\}, \{$

$$\begin{array}{ll} r_1'\colon Ss' \rightarrow Sq_1', & r_3'\colon Sq_1' \rightarrow SAf' \\ r_2'\colon Ss' \rightarrow Sq_2', & r_4'\colon Sq_2' \rightarrow SBf' \end{array}$$

$\}, s', \mathrm{S}, \{f'\})$

and $\Psi = \{ss', q_1q_1', q_2q_2'\}$. This SCAS$_2$, depending on its input, modifies either its first or second pushdown. According to Definition 4.1.3, the allowed combinations of rules are

$$\begin{array}{ll} (r_1, r_1')\colon & \mathrm{card}(\{(r_1, r_1')\}) = 1 \\ (r_2, r_2')\colon & \mathrm{card}(\{(r_2, r_2')\}) = 1 \\ (r_3, r_3')\colon & \mathrm{card}(\{(r_3, r_3')\}) = 1 \\ (r_4, r_4')\colon & \mathrm{card}(\{(r_4, r_4')\}) = 1 \end{array}$$

so Γ is a dSCAS$_2$. Observe that condition $\pi_r(\alpha') \in \Psi_f$ is necessary because its omission leads to

$$\begin{array}{ll} (r_1, r_1')\colon & \mathrm{card}(\{(r_1, r_1'), (r_1, r_2')\}) = 2 \\ (r_2, r_2')\colon & \mathrm{card}(\{(r_2, r_2'), (r_2, r_1')\}) = 2 \\ (r_3, r_3')\colon & \mathrm{card}(\{(r_3, r_3')\}) = 1 \\ (r_4, r_4')\colon & \mathrm{card}(\{(r_4, r_4')\}) = 1 \end{array}$$

which is in contradiction with the fact that Γ is deterministic. $\square$

4.2 Accepting Power

Synchronization by states as defined in Definition 4.1.1 provides a powerful framework for information exchange between components of SCAS$_n$. As will be demonstrated by the following theorems, two pushdown components are sufficient to accept every recursively enumerable language, both in a deterministic and nondeterministic way.

Theorem 4.2.1. *For every recursively enumerable language* L *over an alphabet* Σ*, there exists a deterministic* SCAS$_2$*,* $\Gamma = (\mathrm{M}_1, \mathrm{M}_2, \Psi)$*, where*

$$\mathrm{M}_i = (\mathrm{Q}_i, \Sigma, \Gamma_i, \mathrm{R}_i, s_i, \mathrm{S}_i, \mathrm{F}_i)$$

is a PDA, $1 \leq i \leq 2$*, such that* $\mathrm{L}(\Gamma)_f = \mathrm{L}$.

Proof. Let L be a recursively enumerable language over an alphabet Σ. Then there exists a deterministic two-pushdown automaton, M, such that $\mathrm{L} = \mathrm{L}(\mathrm{M})_f$. Let $\mathrm{M} = (\mathrm{Q}, \Sigma, \hat{\Gamma}, \mathrm{R}, s, \mathrm{S}_1, \mathrm{S}_2, \mathrm{F})$ be a deterministic two-pushdown automaton such that $\mathrm{L} = \mathrm{L}(\mathrm{M})_f$. From M, construct a dSCAS$_2$ Γ in the following way:

1. set $\mathrm{Q}_1 = \emptyset$, $\mathrm{Q}_2 = \emptyset$, and $\Psi = \emptyset$;
2. set $\Gamma_1 = \Gamma_2 = \hat{\Gamma}$;
3. for every rule $r\colon \mathrm{A}\#\mathrm{B}pa \to x\#yq \in \mathrm{R}$:
   - add states $\bar{p}$, $\langle\bar{r}\rangle$, and $\bar{q}$ to Q_1;
   - add states $\hat{p}$, $\langle\hat{r}\rangle$, and $\hat{q}$ to Q_2;

- add rules $A\bar{p}a \rightarrow A\langle\bar{r}\rangle$ and $A\langle\bar{r}\rangle \rightarrow x\bar{q}$ to R_1;
- add rules $B\hat{p} \rightarrow B\langle\hat{r}\rangle$ and $B\langle\hat{r}\rangle \rightarrow y\hat{q}$ to R_2;
- add words $\bar{p}\hat{p}$, $\bar{q}\hat{q}$, and $\langle\bar{r}\rangle\langle\hat{r}\rangle$ to Ψ;

4. set $F_1 = \{\bar{f} \mid f \in F\}$ and $F_2 = \{\hat{f} \mid f \in F\}$.

From the construction above follows that if M is deterministic, then Γ must also be deterministic. The next step serves to prove that $L(M)_f = L(\Gamma)_f$.

Claim 4.2.2. *If* $u_0\#v_0q_0w_0 \vdash^i_M u_i\#v_iq_iw_i$, *then*

$$(u_0\bar{q}_0w_0, v_0\hat{q}_0) \vdash^{2i}_\Gamma (u_i\bar{q}_iw_i, v_i\hat{q}_i)$$

where $u_i \in \Gamma_1^*$, $v_i \in \Gamma_2^*$, $q_i \in Q$, $\bar{q}_i \in Q_1$, $\hat{q}_i \in Q_2$, *and* $w_i \in \Sigma^*$, *for all* $i \geq 0$.

Proof. The proof is established by induction on $i \geq 0$.

Basis. For $i = 0$, $u_0\#v_0q_0w_0 \vdash^0_M u_0\#v_0q_0w_0$ implies that

$$(u_0\bar{q}_0w_0, v_0\hat{q}_0) \vdash^0_\Gamma (u_0\bar{q}_0w_0, v_0\hat{q}_0)$$

Thus, the basis holds.

Induction Hypothesis. Suppose that the claim holds for all $0 \leq i \leq k$, for some $k \geq 0$.

Induction Step. If

$$u_0\#v_0q_0w_0 \vdash^k_M u_k\#v_kq_kw_k \vdash_M u_{k+1}\#v_{k+1}q_{k+1}w_{k+1}$$

then there exists a rule $r : A\#Bq_ka \rightarrow x\#yq_{k+1} \in R$ such that $u_k = \gamma A$, $v_k = \delta B$, $u_{k+1} = \gamma x$, $v_{k+1} = \delta y$, and $w_k = aw_{k+1}$, where $\gamma \in \Gamma_1^*$, $\delta \in \Gamma_2^*$. Then, there also exist rules $A\bar{q}_ka \rightarrow A\langle\bar{r}\rangle \in R_1$, $A\langle\bar{r}\rangle \rightarrow x\bar{q}_{k+1} \in R_1$, $B\hat{q}_k \rightarrow B\langle\hat{r}\rangle \in R_2$, $B\langle\hat{r}\rangle \rightarrow y\hat{q}_{k+1} \in R_2$, and words $\bar{q}_k\hat{q}_k$, $\bar{q}_{k+1}\hat{q}_{k+1}$, $\langle\bar{r}\rangle\langle\hat{r}\rangle \in \Psi$, which implies that

$$\begin{aligned}(u_0\bar{q}_0w_0, v_0\hat{q}_0) &\vdash^{2k}_\Gamma (u_k\bar{q}_kw_k, v_k\hat{q}_k)\\ &\vdash_\Gamma (u_k\langle\bar{r}\rangle w_{k+1}, v_k\langle\hat{r}\rangle)\\ &\vdash_\Gamma (u_{k+1}\bar{q}_{k+1}w_{k+1}, v_{k+1}\hat{q}_{k+1})\end{aligned}$$

and the claim holds for $k + 1$ as well. Therefore, Claim 4.2.2 holds. □

Claim 4.2.3. *If* $(u_0\bar{q}_0w_0, v_0\hat{q}_0) \vdash^{2i}_\Gamma (u_i\bar{q}_iw_i, v_i\hat{q}_i)$, *then*

$$u_0\#v_0q_0w_0 \vdash^i_M u_i\#v_iq_iw_i$$

where $u_i \in \Gamma_1^*$, $v_i \in \Gamma_2^*$, $q_i \in Q$, $\bar{q}_i \in Q_1$, $\hat{q}_i \in Q_2$, *and* $w_i \in \Sigma^*$, *for all* $i \geq 0$.

Proof. The proof is established by induction on $i \geq 0$. Observe that only an even number of computation steps is possible in Γ (see the construction of Γ from M above).

Basis. For $i = 0$

$$(u_0\bar{q}_0w_0, v_0\hat{q}_0) \vdash_\Gamma^0 (u_0\bar{q}_0w_0, v_0\hat{q}_0)$$

implies that $u_0\#v_0q_0w_0 \vdash_M^0 u_0\#v_0q_0w_0$. Thus, the basis holds.

Induction Hypothesis. Suppose that the claim holds for all $0 \leq i \leq k$, for some $k \geq 0$.

Induction Step. If

$$\begin{aligned}(u_0\bar{q}_0w_0, v_0\hat{q}_0) \ &\vdash_\Gamma^{2k} \ (u_k\bar{q}_kw_k, v_k\hat{q}_k)\\ &\vdash_\Gamma \ (u_k\langle\bar{r}\rangle w_{k+1}, v_k\langle\hat{r}\rangle)\\ &\vdash_\Gamma \ (u_{k+1}\bar{q}_{k+1}w_{k+1}, v_{k+1}\hat{q}_{k+1})\end{aligned}$$

then there exist rules $A\bar{q}_ka \rightarrow A\langle\bar{r}\rangle \in R_1$, $A\langle\bar{r}\rangle \rightarrow x\bar{q}_{k+1} \in R_1$, $B\hat{q}_k \rightarrow B\langle\hat{r}\rangle \in R_2$, $B\langle\hat{r}\rangle \rightarrow y\hat{q}_{k+1} \in R_2$, and words $\bar{q}_k\hat{q}_k, \bar{q}_{k+1}\hat{q}_{k+1}, \langle\bar{r}\rangle\langle\hat{r}\rangle \in \Psi$, such that $u_k = \gamma A$, $v_k = \delta B$, $u_{k+1} = \gamma x$, $v_{k+1} = \delta y$, $\gamma \in \Gamma_1^*$, $\delta \in \Gamma_2^*$, $w_k = aw_{k+1}$, and $r : A\#Bq_ka \rightarrow x\#yq_{k+1} \in R$. This implies immediately that

$$u_0\#v_0q_0w_0 \vdash_M^k u_k\#v_kq_kw_k \vdash_M u_{k+1}\#v_{k+1}q_{k+1}w_{k+1}$$

which proves that the claim holds for $k + 1$ as well. Therefore, Claim 4.2.3 holds. □

From Claim 4.2.2 and Claim 4.2.3, it follows immediately that for every $w \in \Sigma^*$

$$S_1\#S_2sw \vdash_M^* \gamma_1\#\gamma_2f \quad \text{if and only if} \quad (S_1s_1w, S_2s_2) \vdash_\Gamma^* (\gamma_1f_1, \gamma_2f_2)$$

where $\gamma_1 \in \Gamma_1^*$, $\gamma_2 \in \Gamma_2^*$. Therefore, $L(M)_f = L(\Gamma)_f$, and Theorem 4.2.1 holds. □

Theorem 4.2.4. *For every recursively enumerable language* L *over an alphabet* Σ, *there exists a deterministic* SCAS$_2$, $\Gamma = (M_1, M_2, \Psi)$, *where*

$$M_i = (Q_i, \Sigma, \Gamma_i, R_i, s_i, S_i, F_i)$$

is a PDA, $1 \leq i \leq 2$, *such that* $L(\Gamma)_\varepsilon = L$.

Proof. Let L be a recursively enumerable language over an alphabet Σ. Then there exists a deterministic two-pushdown automaton, M, such that $L = L(M)_\varepsilon$. The rest of proof is analogous to the proof of Theorem 4.2.1. □

Theorem 4.2.5. *For every recursively enumerable language* L *over an alphabet* Σ, *there exists a deterministic* SCAS$_2$, $\Gamma = (M_1, M_2, \Psi)$, *where*

$$M_i = (Q_i, \Sigma, \Gamma_i, R_i, s_i, S_i, F_i)$$

is a PDA, $1 \leq i \leq 2$*, such that* $L(\Gamma)_{f\varepsilon} = L$.

Proof. The proof is analogous to the proof of Theorem 4.2.4. □

Theorem 4.2.6. *Let* $k \geq 3$. *For every recursively enumerable language* L *over an alphabet* Σ*, there exists a deterministic* $SCAS_k$

$$\Gamma = (M_1, M_2, \ldots, M_k, \Psi)$$

where M_i *is a PDA, for all* $1 \leq i \leq k$*, such that* $L = L(\Gamma)_f$.

Proof. Let L be a recursively enumerable language over an alphabet Σ. Then, by Theorem 4.2.1, there exists a deterministic $SCAS_2$, Γ', such that all its components are PDAs and that $L = L(\Gamma')_f$. Let $\Gamma' = (M'_1, M'_2, \Psi')$ be a deterministic $SCAS_2$, where M'_i is a PDA, $1 \leq i \leq 2$, such that $L = L(\Gamma')_f$. Let $k \geq 3$. From Γ', construct a deterministic $SCAS_k$

$$\Gamma = (M_1, M_2, \ldots, M_k, \Psi)$$

where M_i is a PDA, for all $1 \leq i \leq k$, in the following way:

1. $M_1 = M'_1$, $M_2 = M'_2$;
2. $M_i = (\{s_i\}, \Sigma, \{S_i\}, \{S_i s_i \to S_i s_i\}, s_i, S_i, \{s_i\})$, for all $3 \leq i \leq k$;
3. $\Psi = \Psi'\{s_3 s_4 \ldots s_k\}$.

It is obvious that components M_3 to M_k are redundant in Γ. Therefore, $L(\Gamma)_f = L(\Gamma')_f$, and Theorem 4.2.6 holds. □

Theorem 4.2.7. *Let* $k \geq 3$. *For every recursively enumerable language* L *over an alphabet* Σ*, there exists a deterministic* $SCAS_k$

$$\Gamma = (M_1, M_2, \ldots, M_k, \Psi)$$

where M_i *is a PDA, for all* $1 \leq i \leq k$*, such that* $L = L(\Gamma)_\varepsilon$.

Proof. Let L be a recursively enumerable language over an alphabet Σ. Then, by Theorem 4.2.4, there exists a deterministic $SCAS_2$, Γ', such that all its components are PDAs and $L = L(\Gamma')_\varepsilon$. Let $\Gamma' = (M'_1, M'_2, \Psi')$ be a deterministic $SCAS_2$, where M'_i is a PDA, $1 \leq i \leq 2$, such that $L = L(\Gamma')_\varepsilon$. Let $k \geq 3$. From Γ', construct a deterministic $SCAS_k$

$$\Gamma = (M_1, M_2, \ldots, M_k, \Psi)$$

where M_i is a PDA, for all $1 \leq i \leq k$, in the following way:

1. Let $M'_1 = (Q'_1, \Sigma, \Gamma'_1, R'_1, s'_1, S'_1, F'_1)$. Then $M_1 = (Q_1, \Sigma, \Gamma_1, R_1, s_1, S_1, F'_1)$, where
   - $Q_1 = Q'_1 \cup \{s_1, q_1\}$, where $s_1, q_1 \notin Q'_1$;
   - $\Gamma_1 = \Gamma'_1 \cup \{S_1\}$, where $S_1 \notin \Gamma'_1$;

- $\mathrm{R}_1 = \mathrm{R}_1' \cup \{\mathrm{S}_1 s_1 \to \mathrm{S}_1\mathrm{S}_1' s_1', \mathrm{S}_1 p \to q_1 \mid p \in \mathrm{Q}_1'\}$.

2. Analogously, construct M_2 from M_2'.
3. For all $3 \leq i \leq k$, set $\mathrm{M}_i = (\{s_i, p_i, q_i\}, \Sigma, \{\mathrm{S}_i\}, \{\mathrm{S}_i s_i \to \mathrm{S}_i p_i, \mathrm{S}_i p_i \to \mathrm{S}_i p_i, \mathrm{S}_i p_i \to q_i\}, s_i, \mathrm{S}_i, \emptyset)$.
4. Set $\Psi = \Psi'\{p_3 p_4 \dots p_k\} \cup \{s_1 s_2 \dots s_k, q_1 q_2 \dots q_k\}$.

Γ works in the following way:

1. During its move from s_i to s_i', M_i pushes S_i' on its pushdown, for all $1 \leq i \leq 2$. Simultaneously, M_j moves from s_j to p_j, for all $3 \leq j \leq k$.
2. Γ accepts (or rejects) its input word. During this phase, M_1 and M_2 perform their moves by using the same sequences of rules like M_1' and M_2', respectively. M_3 through M_k loop over states p_3 through p_k, respectively.
3. When M_1' and M_2' in Γ' empty their pushdowns, the pushdowns of M_1 and M_2 in Γ have S_1 and S_2 on their tops, respectively. At this point, M_1 through M_k deterministically empty their pushdowns by moving to states q_1 through q_k, respectively.

Thus, Γ accepts its input word if and only if Γ' accepts its input word, which completes the proof of Theorem 4.2.7. □

Theorem 4.2.8. *Let $k \geq 3$. For every recursively enumerable language* L *over an alphabet* Σ*, there exists a deterministic* SCAS_k

$$\Gamma = (\mathrm{M}_1, \mathrm{M}_2, \dots, \mathrm{M}_k, \Psi)$$

where M_i *is a PDA, for all* $1 \leq i \leq k$*, such that* $\mathrm{L} = \mathrm{L}(\Gamma)_{f\varepsilon}$.

Proof. Prove this by analogy with Theorem 4.2.7 except that states q_1 through q_k are final. □

Corollary 4.2.9. *Let $n \geq 2$. For every recursively enumerable language* L*, there exists a deterministic* SCAS_n*,* Γ*, such that all its components are PDAs and* $\mathrm{L} = \mathrm{L}(\Gamma)_f$.

Proof. This follows from Theorem 4.2.1 and Theorem 4.2.6. □

Corollary 4.2.10. *Let $n \geq 2$. For every recursively enumerable language* L*, there exists a deterministic* SCAS_n*,* Γ*, such that all its components are PDAs and* $\mathrm{L} = \mathrm{L}(\Gamma)_\varepsilon$.

Proof. This follows from Theorem 4.2.4 and Theorem 4.2.7. □

Corollary 4.2.11. *Let $n \geq 2$. For every recursively enumerable language* L*, there exists a deterministic* SCAS_n*,* Γ*, such that all its components are PDAs and* $\mathrm{L} = \mathrm{L}(\Gamma)_{f\varepsilon}$.

Proof. This follows from Theorem 4.2.5 and Theorem 4.2.8. □

The family of recursively enumerable languages can be also characterized by SCAS_n, where $n \geq 2$, such that all its components are one-turn PDAs. However, the question of whether the same holds also for deterministic SCAS_n remains an open problem[10].

Theorem 4.2.12. *Let $x \in \{f, \varepsilon, f\varepsilon\}$. For every recursively enumerable language* L *over an alphabet* Σ*, there exists an* SCAS_2*,* $\Gamma = (\mathrm{M}_1, \mathrm{M}_2, \Psi)$*, where both* M_1 *and* M_2 *are one-turn pushdown automata, such that* $\mathrm{L} = \mathrm{L}(\Gamma)_x$*.*

Proof. Let $x \in \{f, \varepsilon, f\varepsilon\}$. Let L be a recursively enumerable language over an alphabet Σ. By Theorem 2.4.9, there exists a simultaneously one-turn two-pushdown automaton, M, such that $\mathrm{L} = \mathrm{L}(\mathrm{M})_x$. Following Theorem 4.2.1, construct Γ from M. Clearly, following the proof of said theorem, $\mathrm{L}(\Gamma)_f = \mathrm{L}(\mathrm{M})_f$, and as the construction of Γ from M preserves operations on pushdowns, both M_1 and M_2 are one-turn pushdown automata. From Theorem 4.2.4 and Theorem 4.2.5 it follows by analogy that Γ can be constructed from M, where $\mathrm{L}(\Gamma)_\varepsilon = \mathrm{L}(\mathrm{M})_\varepsilon$ and $\mathrm{L}(\Gamma)_{f\varepsilon} = \mathrm{L}(\mathrm{M})_{f\varepsilon}$, respectively, and both M_1 and M_2 are one-turn pushdown automata as well. □

Theorem 4.2.13. *Let $x \in \{f, \varepsilon, f\varepsilon\}$ and $k \geq 3$. For every recursively enumerable language* L *over an alphabet* Σ*, there exists an* SCAS_k*,* $\Gamma = (\mathrm{M}_1, \mathrm{M}_2, \ldots, \mathrm{M}_k, \Psi)$*, where* M_i *is a one-turn PDA, for all* $1 \leq i \leq k$*, such that* $\mathrm{L} = \mathrm{L}(\Gamma)_x$*.*

Proof. Let $x \in \{f, \varepsilon, f\varepsilon\}$ and $k \geq 3$. Let L be a recursively enumerable language over an alphabet Σ. Then, by Theorem 4.2.12, there exists an SCAS_2, $\Gamma' = (\mathrm{M}'_1, \mathrm{M}'_2, \Psi')$, where M'_1 and M'_2 are both one-turn PDAs, such that $\mathrm{L} = \mathrm{L}(\Gamma')_x$. From Γ', construct Γ and then prove the identity of languages accepted by Γ' and Γ in a way analogous to proofs of Theorem 4.2.6, Theorem 4.2.7, and Theorem 4.2.8. Observe that components M_3 through M_k of Γ are all one-turn PDAs. □

Corollary 4.2.14. *Let $x \in \{f, \varepsilon, f\varepsilon\}$ and $n \geq 2$. For every recursively enumerable language* L*, there exists an* SCAS_n*,* Γ*, such that all its components are one-turn PDAs and* $\mathrm{L} = \mathrm{L}(\Gamma)_x$*.*

Proof. This follows from Theorem 4.2.12 and Theorem 4.2.13. □

Open Problem 4.2.15. Let $n \geq 2$. Then, there exists a recursively enumerable language that cannot be accepted by any dSCAS_n such that all its components are one-turn PDAs. □

The previous results are summarized in the following theorem.

Theorem 4.2.16. *For every* $\mathrm{L} \in \mathbf{RE}$*, there exist*

(a) *an* SCAS_n Γ *such that all its components are PDAs and* $\mathrm{L} = \mathrm{L}(\Gamma)_x$*, where* $n \geq 2$ *and* $x \in \{f, \varepsilon, f\varepsilon\}$*;*

(b) *an* dSCAS_n Γ *such that all its components are PDAs and* $\mathrm{L} = \mathrm{L}(\Gamma)_x$*, where* $n \geq 2$ *and* $x \in \{f, \varepsilon, f\varepsilon\}$*;*

[10] The conjecture is that it is not possible because a nondeterminism is a key element here.

(c) an SCAS_n Γ *such that all its components are one-turn PDAs and* $\mathrm{L} = \mathrm{L}(\Gamma)_x$*, where* $n \geq 2$ *and* $x \in \{f, \varepsilon, f\varepsilon\}$.

Proof. This theorem follows from Corollary 4.2.9, Corollary 4.2.10, Corollary 4.2.11, Corollary 4.2.14, and from the obvious observation that deterministic SCAS_ns with two and more pushdown components are no more powerful than their nondeterministic counterparts. □

Next, the case when no more than one pushdown component is permitted in SCAS_n is studied.

Lemma 4.2.17. *Let* $x \in \{f, \varepsilon, f\varepsilon\}$ *and* $n \geq 1$*. For every* SCAS_n

$$\Gamma = (\mathrm{M}_1, \mathrm{M}_2, \ldots, \mathrm{M}_n, \Psi)$$

where for some $1 \leq i \leq n$, M_i *is a PDA, and for all* $1 \leq j \leq n$, $i \neq j$, M_j *is an FA, there exists a PDA,* M, *such that* $\mathrm{L}(\mathrm{M})_x = \mathrm{L}(\Gamma)_x$.

Proof. Let $x \in \{f, \varepsilon, f\varepsilon\}$. For $n = 1$, Γ has only one component, which is a PDA, and therefore Lemma 4.2.17 holds immediately. Below, it is shown that the lemma holds for $n \geq 2$ as well, where only the case that no component of Γ except for the first one can be a PDA is considered (the other situations can be proved analogously).

Let $n \geq 2$ and $\Gamma = (\mathrm{M}_1, \mathrm{M}_2, \ldots, \mathrm{M}_n, \Psi)$ be an SCAS_n, where M_1 is a PDA, and M_2 through M_n are FAs. From Γ, construct a PDA, M, such that $\mathrm{L}(\mathrm{M})_x = \mathrm{L}(\Gamma)_x$, in the following way:

1. Let $\mathrm{M}_1 = (\mathrm{Q}_1, \Sigma, \Gamma_1, \mathrm{R}_1, s_1, \mathrm{S}_1, \mathrm{F}_1)$ be a PDA from Γ, and let

$$\mathrm{M}_i = (\mathrm{Q}_i, \Sigma, \mathrm{R}_i, s_i, \mathrm{F}_i)$$

be an FA from Γ, for all $2 \leq i \leq n$.

2. Set $\mathrm{M} = (\mathrm{Q}, \Sigma, \Gamma_1, \mathrm{R}, \langle s_1 s_2 \ldots s_n \rangle, \mathrm{S}_1, \mathrm{F})$, where

$$\begin{array}{rcl}
\mathrm{Q} & = & \{\langle\omega\rangle \mid \omega \in \Psi \cup \mathrm{F}_1\mathrm{F}_2 \ldots \mathrm{F}_n\} \\
\mathrm{R} & = & \left\{ \mathrm{A}\langle\omega_1\rangle a \rightarrow x\langle\omega_2\rangle \;\middle|\; \begin{array}{l} \mathrm{A}p_1a \rightarrow xq_1 \in \mathrm{R}_1, p_i \rightarrow q_i \in \mathrm{R}_i, 2 \leq i \leq n \\ \omega_1 = p_1p_2 \ldots p_n, \omega_2 = q_1q_2 \ldots q_n \\ \langle\omega_1\rangle, \langle\omega_2\rangle \in \mathrm{Q} \end{array} \right\} \\
\mathrm{F} & = & \{\langle\omega\rangle \mid \omega \in \mathrm{F}_1\mathrm{F}_2 \ldots \mathrm{F}_n\}
\end{array}$$

To prove that $\mathrm{L}(\Gamma)_x = \mathrm{L}(\mathrm{M})_x$, first establish the two following claims.

Claim 4.2.18. *If* $(up_1w, p_2, \ldots, p_n) \vdash_\Gamma^i (u'q_1w', q_2, \ldots, q_n)$*, then*

$$u\langle p_1p_2 \ldots p_n\rangle w \vdash_\mathrm{M}^i u'\langle q_1q_2 \ldots q_n\rangle w'$$

where $u, u' \in \Gamma_1^*$, $w, w' \in \Sigma^*$, $p_i, q_i \in Q_i$, $1 \leq i \leq n$, *and* $\langle p_1 p_2 \dots p_n \rangle, \langle q_1 q_2 \dots q_n \rangle \in Q$.

Proof. The proof is made by induction on $i \geq 0$.

Basis. For $i = 0$, $(up_1w, p_2, \dots, p_n) \vdash_\Gamma^0 (up_1w, p_2, \dots, p_n)$ implies

$$u\langle p_1 p_2 \dots p_n \rangle w \vdash_M^0 u\langle p_1 p_2 \dots p_n \rangle w$$

so the claim holds for $i = 0$.

Induction Hypothesis. Suppose that the claim holds for all $0 \leq i \leq k$, for some $k \geq 0$.

Induction Step. If

$$(uAp_1aw, p_2, \dots, p_n) \vdash_\Gamma (uxo_1w, o_2, \dots, o_n) \vdash_\Gamma^k (u'q_1w', q_2, \dots, q_n)$$

then there exist rules $Ap_1a \to xo_1 \in R_1$, $p_j \to o_j \in R_j$, $2 \leq j \leq n$, and words $p_1p_2 \dots p_n, o_1o_2 \dots o_n \in \Psi \cup F_1F_2 \dots F_n$. According to the construction of M from Γ, this implies that there also exists a rule $A\langle p_1p_2 \dots p_n \rangle a \to x\langle o_1o_2 \dots o_n \rangle \in R$, so

$$uA\langle p_1p_2 \dots p_n \rangle aw \vdash_M ux\langle o_1o_2 \dots o_n \rangle w \vdash_M^k u'\langle q_1q_2 \dots q_n \rangle w'$$

and the claim holds for $k + 1$ as well. Therefore, Claim 4.2.18 holds. □

Claim 4.2.19. *If* $u\langle p_1p_2 \dots p_n \rangle w \vdash_M^i u'\langle q_1q_2 \dots q_n \rangle w'$, *then*

$$(up_1w, p_2, \dots, p_n) \vdash_\Gamma^i (u'q_1w', q_2, \dots, q_n)$$

where $u, u' \in \Gamma_1^*$, $w, w' \in \Sigma^*$, $p_i, q_i \in Q_i$, $1 \leq i \leq n$, *and* $\langle p_1 p_2 \dots p_n \rangle, \langle q_1 q_2 \dots q_n \rangle \in Q$.

Proof. The proof is made by induction on $i \geq 0$.

Basis. For $i = 0$, $u\langle p_1p_2 \dots p_n \rangle w \vdash_M^0 u\langle p_1p_2 \dots p_n \rangle w$ implies

$$(up_1w, p_2, \dots, p_n) \vdash_\Gamma^0 (up_1w, p_2, \dots, p_n)$$

so the claim holds for $i = 0$.

Induction Hypothesis. Suppose that the claim holds for all $0 \leq i \leq k$, for some $k \geq 0$.

Induction Step. If

$$uA\langle p_1p_2 \dots p_n \rangle aw \vdash_M ux\langle o_1o_2 \dots o_n \rangle w \vdash_M^k u'\langle q_1q_2 \dots q_n \rangle w'$$

then there exists a rule $A\langle p_1p_2 \dots p_n \rangle a \to x\langle o_1o_2 \dots o_n \rangle \in R$, which implies that there must exist words $p_1p_2 \dots p_n, o_1o_2 \dots o_n \in \Psi \cup F_1F_2 \dots F_n$, and rules $Ap_1a \to xo_1 \in R_1$, $p_j \to o_j \in R_j$, $2 \leq j \leq n$, in Γ, so

$$(uAp_1aw, p_2, \dots, p_n) \vdash_\Gamma (uxo_1w, o_2, \dots, o_n) \vdash_\Gamma^k (u'q_1w', q_2, \dots, q_n)$$

and the claim holds for $k + 1$ as well. Therefore, Claim 4.2.19 holds. □

From Claim 4.2.18 and Claim 4.2.19, it follows immediately that for every $w \in \Sigma^*$

1. $(S_1 s_1 w, s_2, \dots, s_n) \vdash_\Gamma^* (\gamma f_1, f_2 \dots, f_n)$ if and only if $S_1 \langle s_1 s_2 \dots s_n \rangle w \vdash_M^* \gamma \langle f_1 f_2 \dots f_n \rangle$
2. $(S_1 s_1 w, s_2, \dots, s_n) \vdash_\Gamma^* (q_1, q_2, \dots, q_n)$ if and only if $S_1 \langle s_1 s_2 \dots s_n \rangle w \vdash_M^* \langle q_1 q_2 \dots q_n \rangle$
3. $(S_1 s_1 w, s_2, \dots, s_n) \vdash_\Gamma^* (f_1, f_2, \dots, f_n)$ if and only if $S_1 \langle s_1 s_2 \dots s_n \rangle w \vdash_M^* \langle f_1 f_2 \dots f_n \rangle$

where $\gamma \in \Gamma_1^*$, $q_i \in Q_i$, $f_i \in F_i$, $1 \leq i \leq n$. Therefore, $L(M)_x = L(\Gamma)_x$, and Lemma 4.2.17 holds. □

Lemma 4.2.20. *Let $x \in \{f, \varepsilon, f\varepsilon\}$ and $n \geq 1$. For every PDA, M, there exists an $SCAS_n$, $\Gamma = (M_1, M_2, \dots, M_n, \Psi)$, where M_1 is a PDA, and M_2 through M_n are FAs, such that $L(\Gamma)_x = L(M)_x$.*

Proof. Let $x \in \{f, \varepsilon, f\varepsilon\}$. Let $M = (Q, \Sigma, \Gamma_M, R, s, S, F)$ be a PDA. For $n = 1$, $\Gamma = (M, Q)$, and the lemma holds immediately. For some $n \geq 2$, construct an $SCAS_n$

$$\Gamma = (M_1, M_2, \dots, M_n, \Psi)$$

where M_1 is a PDA, and M_i is an FA, for all $2 \leq i \leq n$, such that $L(\Gamma)_x = L(M)_x$, in the following way:

1. Set $M_1 = M$.
2. For every $2 \leq i \leq n$, set $M_i = (\{s_i\}, \Sigma, \{s_i \to s_i\}, s_i, \{s_i\})$.
3. Set $\Psi = Q\{s_2 s_3 \dots s_n\}$.

Thus, for every $w \in \Sigma^*$

1. $(Ssw, s_2, \dots, s_n) \vdash_\Gamma^* (\gamma f, s_2, \dots, s_n)$ if and only if $Ssw \vdash_M^* \gamma f$
2. $(Ssw, s_2, \dots, s_n) \vdash_\Gamma^* (q, s_2, \dots, s_n)$ if and only if $Ssw \vdash_M^* q$
3. $(Ssw, s_2, \dots, s_n) \vdash_\Gamma^* (f, s_2, \dots, s_n)$ if and only if $Ssw \vdash_M^* f$

where $\gamma \in \Gamma_M^*$, $q \in Q$, $f \in F$. Therefore, $L(\Gamma)_x = L(M)_x$, and Lemma 4.2.20 holds. □

Theorem 4.2.21. *For every $L \in$ **CF**, there is an $SCAS_n$ Γ containing a PDA and $n - 1$ FAs as its components such that $L = L(\Gamma)_x$, where $n \geq 1$ and $x \in \{f, \varepsilon, f\varepsilon\}$.*

Proof. This theorem follows from Lemma 4.2.17 and Lemma 4.2.20. □

Theorem 4.2.22. *For every $L \in$ **LIN**, there is an $SCAS_n$ Γ containing a one-turn PDA and $n - 1$ FAs as its components such that $L = L(\Gamma)_x$, where $n \geq 1$ and $x \in \{f, \varepsilon, f\varepsilon\}$.*

Proof. Let $L \in$ **LIN** and $n \geq 1$. Let $x \in \{f, \varepsilon, f\varepsilon\}$. Then, there exists a one-turn PDA, M, such that L is accepted by M. From M, as is demonstrated in Lemma 4.2.20, construct an

SCAS$_n$, Γ, such that $\mathrm{L(M)}_x = \mathrm{L}(\Gamma)_x$. Thus, Γ has only one PDA as its component, which must be a one-turn PDA.

Conversely, let Γ be an SCAS$_n$ with at most one one-turn PDA and arbitrary number of FAs as its components. Let $x \in \{f, \varepsilon, f\varepsilon\}$. By Lemma 4.2.17, there exists a PDA, M, such that $\mathrm{L}(\Gamma)_x = \mathrm{L(M)}_x$. By Definition 4.1.1, if it is possible to perform only one turn in a PDA component of Γ, then no more than one turn is possible to perform in Γ. Therefore, from the construction of M from Γ in Lemma 4.2.17, it follows that M must be a one-turn PDA, which implies that $\mathrm{L(M)}_x \in$ **LIN**. □

Corollary 4.2.23. *Let $x \in \{f, \varepsilon, f\varepsilon\}$ and let $n \geq 1$. For every* $\mathrm{L} \in \mathscr{L}(\mathrm{dPDA})_x$*, there is an* dSCAS$_n$*,* Γ*, containing a PDA and $n - 1$ FAs as its components such that* $\mathrm{L} = \mathrm{L}(\Gamma)_x$.

Proof. It directly follows from Lemma 4.2.17 and Lemma 4.2.20 (follow their proofs for deterministic variants of SCAS$_n$s and PDAs). □

Chapter 5
Unlimited Deep Pushdown Automata

In [68], Meduna introduced a new language model, called deep pushdown automaton, which works like pushdown automaton but allows expansion of pushdown symbols deeper on the pushdown up to a given limit. As recalled in Chapter 3, such modified pushdown automata are equivalent to k-limited state grammars. A natural question concerning the limit becomes apparent: how does the accepting power of deep pushdown automata change if there is no limit imposed on the depth of expansion of the pushdown?

The present chapter introduces such a kind of automata, called *unlimited deep pushdown automata*, and studies their accepting power. Section 5.1 gives a definition and example of *absolutely unlimited deep pushdown automata*, published in [54], and also gives the formal proofs of all stated theorems about their accepting power. Section 5.2 does the same for *relatively unlimited deep pushdown automata*, which have not yet been published.

5.1 Absolutely Unlimited Deep Pushdown Automata

Recall the definition of unlimited deep pushdown automata from [54]. Informally, during every move, an absolutely unlimited deep pushdown automaton either pops or expands its pushdown. In case the topmost pushdown symbol is a symbol from the input alphabet, it is compared with the current input symbol and if they correspond, the pushdown symbol is popped and the input symbol is read. Otherwise, the pushdown may be expanded. With absolutely unlimited deep pushdown expansion, an expandable pushdown symbol is chosen and its topmost occurrence is rewritten.

Definition 5.1.1. An *absolutely unlimited deep pushdown automaton* (abbreviated AUD-PDA), M, is an 8-tuple $M = (Q, \Sigma, \Gamma, \#, R, s, S, F)$, where

- Q is a finite set of *states*;
- Σ is an *input alphabet*;
- Γ is a *pushdown alphabet*, $\Gamma \cap Q = \emptyset$, $\Sigma \subset \Gamma$;
- $\# \in (\Gamma - \Sigma)$ is the special symbol called *bottom symbol*;

- $\mathrm{R} \subseteq (\mathrm{Q} \times (\Gamma - (\Sigma \cup \{\#\})) \times \mathrm{Q} \times (\Gamma - \{\#\})^*) \cup (\mathrm{Q} \times \{\#\} \times \mathrm{Q} \times (\Gamma - \{\#\})^* \{\#\})$ is a finite set of *rules*;
- $s \in \mathrm{Q}$ is the *initial state*;
- $\mathrm{S} \in \Gamma$ is the *initial pushdown symbol*;
- $\mathrm{F} \subseteq \mathrm{Q}$ is the set of *final states*.

A rule $(p, \mathrm{A}, q, x) \in \mathrm{R}$ is usually written as $p\mathrm{A} \rightarrow qx \in \mathrm{R}$. M is said to be *propagating* if $p\mathrm{A} \rightarrow qx \in \mathrm{R}$ implies $x \neq \varepsilon$.

Set $\Xi = \mathrm{Q} \times \Sigma^* \times (\Gamma - \{\#\})^* \{\#\}$. Then, $\chi \in \Xi$ is said to be a *configuration* of M.

The M-based relation of *direct pop move*, ${}_{p}^{a}\vdash_{\mathrm{M}}$, over Ξ is defined as follows:

$$(p, aw, az) \; {}_{p}^{a}\vdash_{\mathrm{M}} \; (p, w, z)$$

where $p \in \mathrm{Q}$, $a \in \Sigma$, $w \in \Sigma^*$, and $z \in \Gamma^*$.

The M-based relation of *direct expansion move*, ${}_{e}^{a}\vdash_{\mathrm{M}}$, over Ξ is defined as follows:

$$(p, w, u\mathrm{A}v) \; {}_{e}^{a}\vdash_{\mathrm{M}} \; (q, w, uxv)$$

if and only if $p\mathrm{A} \rightarrow qx \in \mathrm{R}$, $\mathrm{A} \notin \mathrm{alph}(u)$ and for every $\mathrm{A}' \in (\mathrm{alph}(u) - \Sigma)$, $p\mathrm{A}' \rightarrow q'x' \notin \mathrm{R}$, where $p, q, q' \in \mathrm{Q}$, $w \in \Sigma^*$, $\mathrm{A} \in (\Gamma - \Sigma)$, and $u, v, x, x' \in \Gamma^*$.

The M-based relation of *direct move*, ${}^{a}\vdash_{\mathrm{M}}$, is then defined as ${}^{a}\vdash_{\mathrm{M}} = {}_{p}^{a}\vdash_{\mathrm{M}} \cup {}_{e}^{a}\vdash_{\mathrm{M}}$. For $k \geq 0$, ${}_{p}^{a}\vdash_{\mathrm{M}}^{k}$, ${}_{e}^{a}\vdash_{\mathrm{M}}^{k}$, ${}^{a}\vdash_{\mathrm{M}}^{k}$, ${}_{p}^{a}\vdash_{\mathrm{M}}^{+}$, ${}_{e}^{a}\vdash_{\mathrm{M}}^{+}$, ${}^{a}\vdash_{\mathrm{M}}^{+}$, ${}_{p}^{a}\vdash_{\mathrm{M}}^{*}$, ${}_{e}^{a}\vdash_{\mathrm{M}}^{*}$, and ${}^{a}\vdash_{\mathrm{M}}^{*}$ are defined as usual.

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

$$\mathrm{L(M)} = \{w \mid (s, w, \mathrm{S}\#) \; {}^{a}\vdash_{\mathrm{M}}^{*} \; (f, \varepsilon, \#), w \in \Sigma^*, f \in \mathrm{F}\}$$

Additionally, the language accepted by M by empty pushdown, $\mathrm{L(M)}_{\varepsilon}$, is defined as

$$\mathrm{L(M)}_{\varepsilon} = \{w \mid (s, w, \mathrm{S}\#) \; {}^{a}\vdash_{\mathrm{M}}^{*} \; (q, \varepsilon, \#), w \in \Sigma^*, q \in \mathrm{Q}\}$$

AUDPDA and **AUDPDA**$^{-\varepsilon}$ denote the families of languages accepted by AUDPDA and propagating AUDPDA, respectively. □

In the following example, it is demonstrated that absolutely unlimited deep pushdown automata are capable of accepting languages that are not context-free.

Example 5.1.2. Consider the absolutely unlimited deep pushdown automaton

$$\mathrm{M} = (\mathrm{Q}, \{a\}, \{\mathrm{S}, \mathrm{A}, \mathrm{X}, \mathrm{A}', \mathrm{X}', \#, a\}, \#, \mathrm{R}, \langle s \rangle, \mathrm{S}, \{\langle f \rangle\})$$

with $\mathrm{Q} = \{\langle s \rangle, \langle c \rangle, \langle 1 \rangle, \langle 2 \rangle, \langle 1' \rangle, \langle 2' \rangle, \langle f \rangle\}$ and R containing rules

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\langle s \rangle\mathrm{S} & \rightarrow & \langle c \rangle a\mathrm{SAX} & \langle c \rangle\mathrm{S} & \rightarrow & \langle c \rangle a\mathrm{SA} \\
\langle c \rangle\mathrm{S} & \rightarrow & \langle 1 \rangle & \langle 1 \rangle\mathrm{A} & \rightarrow & \langle 2 \rangle \\
\langle 1 \rangle\mathrm{X} & \rightarrow & \langle 1' \rangle\mathrm{X}' & \langle 2 \rangle\mathrm{A} & \rightarrow & \langle 1 \rangle\mathrm{A}' \\
\langle 2 \rangle\mathrm{X} & \rightarrow & \langle f \rangle & \langle 1' \rangle\mathrm{A}' & \rightarrow & \langle 2' \rangle \\
\langle 1' \rangle\mathrm{X}' & \rightarrow & \langle 1 \rangle\mathrm{X} & \langle 2' \rangle\mathrm{A}' & \rightarrow & \langle 1' \rangle\mathrm{A} \\
\langle 2' \rangle\mathrm{X}' & \rightarrow & \langle f \rangle & & &
\end{array}$$

Then a word $aaaa$ is accepted by M in the following way:

$$
\begin{array}{lcll}
(\langle s\rangle, aaaa, \mathrm{S\#}) & {}^{a}_{e}\vdash & (\langle c\rangle, aaaa, a\mathrm{SAX\#}) & [\langle s\rangle\mathrm{S} \to \langle c\rangle a\mathrm{SAX}] \\
 & {}^{a}_{p}\vdash & (\langle c\rangle, aaa, \mathrm{SAX\#}) & \\
 & {}^{a}_{e}\vdash & (\langle c\rangle, aaa, a\mathrm{SAAX\#}) & [\langle c\rangle\mathrm{S} \to \langle c\rangle a\mathrm{SA}] \\
 & {}^{a}_{p}\vdash & (\langle c\rangle, aa, \mathrm{SAAX\#}) & \\
 & {}^{a}_{e}\vdash & (\langle c\rangle, aa, a\mathrm{SAAAX\#}) & [\langle c\rangle\mathrm{S} \to \langle c\rangle a\mathrm{SA}] \\
 & {}^{a}_{p}\vdash & (\langle c\rangle, a, \mathrm{SAAAX\#}) & \\
 & {}^{a}_{e}\vdash & (\langle c\rangle, a, a\mathrm{SAAAAX\#}) & [\langle c\rangle\mathrm{S} \to \langle c\rangle a\mathrm{SA}] \\
 & {}^{a}_{p}\vdash & (\langle c\rangle, \varepsilon, \mathrm{SAAAAX\#}) & \\
 & {}^{a}_{e}\vdash & (\langle 1\rangle, \varepsilon, \mathrm{AAAAX\#}) & [\langle c\rangle\mathrm{S} \to \langle 1\rangle] \\
 & {}^{a}_{e}\vdash & (\langle 2\rangle, \varepsilon, \mathrm{AAAX\#}) & [\langle 1\rangle\mathrm{A} \to \langle 2\rangle] \\
 & {}^{a}_{e}\vdash & (\langle 1\rangle, \varepsilon, \mathrm{A'AAX\#}) & [\langle 2\rangle\mathrm{A} \to \langle 1\rangle\mathrm{A'}] \\
 & {}^{a}_{e}\vdash & (\langle 2\rangle, \varepsilon, \mathrm{A'AX\#}) & [\langle 1\rangle\mathrm{A} \to \langle 2\rangle] \\
 & {}^{a}_{e}\vdash & (\langle 1\rangle, \varepsilon, \mathrm{A'A'X\#}) & [\langle 2\rangle\mathrm{A} \to \langle 1\rangle\mathrm{A'}] \\
 & {}^{a}_{e}\vdash & (\langle 1'\rangle, \varepsilon, \mathrm{A'A'X'\#}) & [\langle 1\rangle\mathrm{X} \to \langle 1'\rangle\mathrm{X'}] \\
 & {}^{a}_{e}\vdash & (\langle 2'\rangle, \varepsilon, \mathrm{A'X'\#}) & [\langle 1'\rangle\mathrm{A'} \to \langle 2'\rangle] \\
 & {}^{a}_{e}\vdash & (\langle 1'\rangle, \varepsilon, \mathrm{AX'\#}) & [\langle 2'\rangle\mathrm{A'} \to \langle 1'\rangle\mathrm{A}] \\
 & {}^{a}_{e}\vdash & (\langle 1\rangle, \varepsilon, \mathrm{AX\#}) & [\langle 1'\rangle\mathrm{X'} \to \langle 1\rangle\mathrm{X}] \\
 & {}^{a}_{e}\vdash & (\langle 2\rangle, \varepsilon, \mathrm{X\#}) & [\langle 1\rangle\mathrm{A} \to \langle 2\rangle] \\
 & {}^{a}_{e}\vdash & (\langle f\rangle, \varepsilon, \mathrm{\#}) & [\langle 2\rangle\mathrm{X} \to \langle f\rangle]
\end{array}
$$

In brief, $(\langle s\rangle, aaaa, \mathrm{S\#})\ {}^{a}\vdash^{*} (\langle f\rangle, \varepsilon, \#)$. Observe that $\mathrm{L(M)} = \mathrm{L(M)}_{\varepsilon} = \{a^{2^n} \mid n \geq 0\}$, which belongs to **CS** − **CF**. □

Accepting Power

Example 5.1.2 demonstrates that unlimited deep pushdown automata are capable of accepting languages that are not semilinear. Hence, unlimited deep pushdown automata are obviously stronger than deep pushdown automata with a finite depth of expansion. Moreover, as shown next, they are capable of accepting every recursively enumerable language.

Lemma 5.1.3. *For every (propagating) state grammar,* G, *there exists a (propagating) absolutely unlimited deep pushdown automaton,* M, *such that* L(G) = L(M).

Proof. Let $\mathrm{G} = (\mathrm{V}, \mathrm{T}, \mathrm{K}, \mathrm{P}, \mathrm{S}, s)$ be a state grammar. Set $\mathrm{N} = (\mathrm{V} - \mathrm{T})$. Introduce the AUDPDA $\mathrm{M} = (\mathrm{K} \cup \{\bar{f}\}, \mathrm{T}, \mathrm{V} \cup \{\#\}, \#, \mathrm{R}, s, \mathrm{S}, \{\bar{f}\})$, where R is constructed by performing the following steps:

(i) for every $(\mathrm{A}, p) \to (x, q) \in \mathrm{P}$, where $p, q \in \mathrm{K}$, $\mathrm{A} \in \mathrm{N}$, and $x \in \mathrm{V}^*$, add $p\mathrm{A} \to qx$ to R;

(ii) for every $p \in \mathrm{K}$, add $p\# \to \bar{f}\#$ to R.

Claim 5.1.4. *Let* $(\mathrm{S}, p) \Rightarrow_{\mathrm{G}}^{m} (wy, q)$*, where* $p, q \in \mathrm{K}$*,* $w \in \mathrm{T}^*$*,* $y \in (\mathrm{NT}^*)^*$*, and* $m \geq 0$*. Then,* $(p, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{*} (q, \varepsilon, y\#)$*.*

Proof. This claim is proved by induction on $m \geq 0$.

Basis. Let $m = 0$, so $(\mathrm{S}, p) \Rightarrow_{\mathrm{G}}^{0} (\mathrm{S}, p)$, $w = \varepsilon$ and $y = \mathrm{S}$. Then

$$(p, \varepsilon, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{0} (p, \varepsilon, \mathrm{S}\#)$$

so the basis holds.

Induction Hypothesis. Assume that the claim holds for all $0 \leq m \leq k$, where k is a non-negative integer.

Induction Step. Let $(\mathrm{S}, p) \Rightarrow_{\mathrm{G}}^{k+1} (wy, q)$, where $p, q \in \mathrm{K}$, $w \in \mathrm{T}^*$, and $y \in (\mathrm{NT}^*)^*$. Since $k + 1 \geq 1$, express $(\mathrm{S}, p) \Rightarrow_{\mathrm{G}}^{k+1} (wy, q)$ as $(\mathrm{S}, p) \Rightarrow_{\mathrm{G}}^{k} (w'u\mathrm{A}v, t) \Rightarrow_{\mathrm{G}} (w'uxv, q)\,[(\mathrm{A}, t) \rightarrow (x, q)]$, where $t \in \mathrm{K}$, $w' \in \mathrm{T}^*$, $u \in (\mathrm{NT}^*)^*$, $\mathrm{A} \in \mathrm{N}$, $x, v \in \mathrm{V}^*$, $(\mathrm{A}, t) \rightarrow (x, q) \in \mathrm{P}$, $w = w'\hat{w}$, and $\hat{w}y = uxv$ with $\hat{w} \in \mathrm{T}^*$. By the induction hypothesis, there exists a move $(p, w', \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{*} (t, \varepsilon, u\mathrm{A}v\#)$, which implies that there also exists a move $(p, w'\hat{w}, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{*} (t, \hat{w}, u\mathrm{A}v\#)$. By the definition of a G-based relation of direct derivation, there is no other rule $r_{\mathrm{P}} \in \mathrm{P}$ such that $\mathrm{lhs}(r_{\mathrm{P}}) = (\mathrm{A}', t)$, for all $\mathrm{A}' \in (\mathrm{alph}(u) - \mathrm{T})$. From the first step of the construction of R follows that there must be a rule $t\mathrm{A} \rightarrow qx \in \mathrm{R}$. Thus, $(t, \hat{w}, u\mathrm{A}v\#)\ {}^{a}_{e}\!\vdash_{\mathrm{M}} (q, \hat{w}, uxv\#)\,[t\mathrm{A} \rightarrow qx]$ and there is no other rule $r_{\mathrm{R}} \in \mathrm{R}$ such that $\mathrm{lhs}(r_{\mathrm{R}}) = t\mathrm{A}'$, for all $\mathrm{A}' \in (\mathrm{alph}(u) - \mathrm{T})$. Since $\hat{w}y = uxv$, it holds that $(q, \hat{w}, \hat{w}y\#)\ {}^{a}_{p}\!\vdash_{\mathrm{M}}^{|\hat{w}|} (q, \varepsilon, y\#)$, which completes the induction step. □

By the previous claim for $y = \varepsilon$, if $(\mathrm{S}, s) \Rightarrow_{\mathrm{G}}^{*} (w, q)$, where $q \in \mathrm{K}$ and $w \in \mathrm{T}^*$, then $(s, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{*} (q, \varepsilon, \#)$. Since $q\# \rightarrow \bar{f}\# \in \mathrm{R}$, it also holds that $(s, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{*} (\bar{f}, \varepsilon, \#)$. Thus, $w \in \mathrm{L(G)}$ implies $w \in \mathrm{L(M)}$, so $\mathrm{L(G)} \subseteq \mathrm{L(M)}$.

Claim 5.1.5. *Let* $(p, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{m} (q, \varepsilon, \hat{w}y\#)$*, where* $p, q \in \mathrm{K}$*,* $w, \hat{w} \in \mathrm{T}^*$*,* $y \in (\mathrm{NT}^*)^*$*, and* $m \geq 0$*. Then,* $(\mathrm{S}, p) \Rightarrow_{\mathrm{G}}^{*} (w\hat{w}y, q)$*.*

Proof. This claim is proved by induction on $m \geq 0$.

Basis. Let $m = 0$. Then, $w = \hat{w} = \varepsilon$, $y = \mathrm{S}$, and

$$(p, \varepsilon, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{0} (p, \varepsilon, \mathrm{S}\#)$$

As $(\mathrm{S}, p) \Rightarrow_{\mathrm{G}}^{0} (\mathrm{S}, p)$, the basis holds.

Induction Hypothesis. Assume that the claim holds for all $0 \leq m \leq k$, where k is a non-negative integer.

Induction Step. Let $(p, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{k+1} (q, \varepsilon, \hat{w}y\#)$, where $p, q \in \mathrm{K}$, $w, \hat{w} \in \mathrm{T}^*$, and $y \in (\mathrm{NT}^*)^*$. Since $k + 1 \geq 1$, express $(p, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{k+1} (q, \varepsilon, \hat{w}y\#)$ as

$$(p, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{k} \chi\ {}^{a}\!\vdash_{\mathrm{M}} (q, \varepsilon, \hat{w}y\#)$$

where χ is a configuration of M whose form depends on whether the last move is a popping move or an expansion.

(I) Assume that $\chi\ {}_{p}^{a}\!\vdash_{\mathrm{M}} (q, \varepsilon, \hat{w}y\#)$. In the greater detail, let

$$\chi = (q, a, a\hat{w}y\#)$$

with $a \in \mathrm{T}$ such that $w = w'a$, where $w' \in \mathrm{T}^*$. Thus

$$(p, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{k} (q, a, a\hat{w}y\#)\ {}_{p}^{a}\!\vdash_{\mathrm{M}} (q, \varepsilon, \hat{w}y\#)$$

Since $(p, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{k} (q, a, a\hat{w}y\#)$, it holds that

$$(p, w', \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{k} (q, \varepsilon, a\hat{w}y\#)$$

By the induction hypothesis, $(\mathrm{S}, p) \Rightarrow_{\mathrm{G}}^{*} (w'a\hat{w}y, q)$. As $w = w'a$, $(\mathrm{S}, p) \Rightarrow_{\mathrm{G}}^{*} (w\hat{w}y, q)$.

(II) Assume that $\chi\ {}_{e}^{a}\!\vdash_{\mathrm{M}} (q, \varepsilon, \hat{w}y\#)$. If this expansion is made by the rule introduced in step (ii), then $q = \bar{f}$, $\hat{w} = \varepsilon$, $y = \varepsilon$, and the induction step follows from the induction hypothesis. Therefore, suppose that this expansion is made by a rule introduced in step (i). In greater detail, suppose that $\chi = (t, \varepsilon, u\mathrm{A}v\#)$ and

$$(t, \varepsilon, u\mathrm{A}v\#)\ {}_{e}^{a}\!\vdash_{\mathrm{M}} (q, \varepsilon, uxv\#)$$

by using $t\mathrm{A} \rightarrow qx \in \mathrm{R}$, where $t \in \mathrm{K}$, $\mathrm{A} \in \mathrm{N}$, $u \in (\mathrm{NT}^*)^*$, $v, x \in \mathrm{V}^*$, and $\hat{w}y = uxv$. By the induction hypothesis, $(p, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{k} (t, \varepsilon, u\mathrm{A}v\#)$ implies $(\mathrm{S}, p) \Rightarrow_{\mathrm{G}}^{*} (wu\mathrm{A}v, t)$. As $t\mathrm{A} \rightarrow qx \in \mathrm{R}$, $(\mathrm{A}, t) \rightarrow (x, q) \in \mathrm{P}$ and for every $\mathrm{A}' \in (\mathrm{alph}(u) - \mathrm{T})$, there is no other rule $r_{\mathrm{P}} \in \mathrm{P}$ such that $\mathrm{lhs}(r_{\mathrm{P}}) = (\mathrm{A}', t)$. Thus, $(\mathrm{S}, p) \Rightarrow_{\mathrm{G}}^{*} (wu\mathrm{A}v, t) \Rightarrow_{\mathrm{G}} (wuxv, q)$. Thus, $(\mathrm{S}, p) \Rightarrow_{\mathrm{G}}^{*} (w\hat{w}y, q)$ since $\hat{w}y = uxv$. □

Consider the previous claim for $\hat{w} = y = \varepsilon$ to see that

$$(s, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{*} (q, \varepsilon, \#)$$

implies $(\mathrm{S}, s) \Rightarrow_{\mathrm{G}}^{*} (w, q)$. Let $w \in \mathrm{L(M)}$. Then

$$(s, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{*} (\bar{f}, \varepsilon, \#)$$

can be expressed as $(s, w, \mathrm{S}\#)\ {}^{a}\!\vdash_{\mathrm{M}}^{*} (q, \varepsilon, \#)\ {}_{e}^{a}\!\vdash_{\mathrm{M}} (\bar{f}, \varepsilon, \#)$. Observe that the last move is made by a rule introduced in step (ii). By the previous claim, $(\mathrm{S}, s) \Rightarrow_{\mathrm{G}}^{*} (w, q)$, so $w \in \mathrm{L(G)}$. Thus, $w \in \mathrm{L(M)}$ implies $w \in \mathrm{L(G)}$, so $\mathrm{L(M)} \subseteq \mathrm{L(G)}$.

As $\mathrm{L(M)} \subseteq \mathrm{L(G)}$ and $\mathrm{L(G)} \subseteq \mathrm{L(M)}$, $\mathrm{L(G)} = \mathrm{L(M)}$. Thus, Lemma 5.1.3 holds. Since the construction of M from G preserves the propagating property, the lemma also holds for propagating state grammars and absolutely unlimited deep pushdown automata. □

Lemma 5.1.6. *For every (propagating) absolutely unlimited deep pushdown automaton,* M, *there exists a (propagating) state grammar,* G, *such that* $\mathrm{L(M)}\{\flat\} = \mathrm{L(G)}$, *where* $\flat$ *is a new symbol such that* $\flat \notin \bigcup_{x \in \mathrm{L(M)}} \mathrm{alph}(x)$.

Proof. Let $\mathrm{M} = (\mathrm{Q}, \Sigma, \Gamma, \#, \mathrm{R}, s, \mathrm{S}, \mathrm{F})$ be an absolutely unlimited deep pushdown automaton. Set $\mathrm{N} = (\Gamma - \Sigma)$. Introduce the state grammar, $\mathrm{G} = (\Gamma \cup \{\mathrm{Z}, \flat\}, \Sigma \cup \{\flat\}, \mathrm{K}, \mathrm{P}, \bar{s}, \mathrm{Z})$, where

$$\mathrm{K} = \mathrm{Q} \cup \{\bar{s}, \bar{f}\}$$

and P is constructed by performing the following steps:

(i) add $(\mathrm{Z}, \bar{s}) \rightarrow (\mathrm{S}\#, s)$ to P;

(ii) for every $p\mathrm{A} \rightarrow qx \in \mathrm{R}$, where $p, q \in \mathrm{Q}$, $\mathrm{A} \in \mathrm{N}$, and $x \in \Gamma^*$, add $(\mathrm{A}, p) \rightarrow (x, q)$ to P;

(iii) for every $p \in \mathrm{Q}$, add $(\#, p) \rightarrow (\flat, \bar{f})$ to P.

Claim 5.1.7. *Let* $(\mathrm{S}\#, p) \Rightarrow_{\mathrm{G}}^{m} (wy\#, q)$*, where* $p, q \in \mathrm{Q}$*,* $w \in \Sigma^*$*,* $y \in ((\mathrm{N} - \{\#\})\Sigma^*)^*$*, and* $m \geq 0$*. Then,* $(p, w, \mathrm{S}\#)\ {}_{a}\!\vdash_{\mathrm{M}}^{*} (q, \varepsilon, y\#)$*.*

Proof. This claim is proved by induction on $m \geq 0$.

Basis. Let $m = 0$, so $(\mathrm{S}\#, p) \Rightarrow_{\mathrm{G}}^{0} (\mathrm{S}\#, p)$, $w = \varepsilon$ and $y = \mathrm{S}$. Then $(p, \varepsilon, \mathrm{S}\#)\ {}_{a}\!\vdash_{\mathrm{M}}^{0} (p, \varepsilon, \mathrm{S}\#)$, so the basis holds.

Induction Hypothesis. Assume that the claim holds for all $0 \leq m \leq k$, where k is a non-negative integer.

Induction Step. Let $(\mathrm{S}\#, p) \Rightarrow_{\mathrm{G}}^{k+1} (wy\#, q)$, where $p, q \in \mathrm{Q}$, $w \in \Sigma^*$, and $y \in ((\mathrm{N}-\{\#\})\Sigma^*)^*$. Observe that rules introduced in steps (i) and (iii) are not used.
Since $k + 1 \geq 1$, express $(\mathrm{S}\#, p) \Rightarrow_{\mathrm{G}}^{k+1} (wy\#, q)$ as

$$(\mathrm{S}\#, p) \Rightarrow_{\mathrm{G}}^{k} (w'u\mathrm{A}v, t) \Rightarrow_{\mathrm{G}} (w'uxv, q)\,[(\mathrm{A}, t) \rightarrow (x, q)]$$

where $t \in \mathrm{Q}$, $w' \in \Sigma^*$, $u \in ((\mathrm{N} - \{\#\})\Sigma^*)^*$, $\mathrm{A} \in \mathrm{N}$, $x, v \in \Gamma^*$, $(\mathrm{A}, t) \rightarrow (x, q) \in \mathrm{P}$, and $wy\# = w'uxv$. Express w as $w = w'\hat{w}$ with $\hat{w} \in \Sigma^*$, so $\hat{w}y\# = uxv$. By the induction hypothesis, $(\mathrm{S}\#, p) \Rightarrow_{\mathrm{G}}^{k} (w'u\mathrm{A}v, t)$ implies

$$(p, w', \mathrm{S}\#)\ {}_{a}\!\vdash_{\mathrm{M}}^{*} (t, \varepsilon, u\mathrm{A}v)$$

which implies $(p, w'\hat{w}, \mathrm{S}\#)\ {}_{a}\!\vdash_{\mathrm{M}}^{*} (t, \hat{w}, u\mathrm{A}v)$. As $(\mathrm{A}, t) \rightarrow (x, q) \in \mathrm{P}$ and there is no other rule $r_{\mathrm{P}} \in \mathrm{P}$ such that $\mathrm{lhs}(r_{\mathrm{P}}) = (\mathrm{A}', t)$, for all $\mathrm{A}' \in (\mathrm{alph}(u) - \Sigma)$, $t\mathrm{A} \rightarrow qx \in \mathrm{R}$ must be the only applicable rule on $(t, \hat{w}, u\mathrm{A}v)$. Thus, $(t, \hat{w}, u\mathrm{A}v)\ {}_{e}^{a}\!\vdash_{\mathrm{M}} (q, \hat{w}, uxv)$. Since $\hat{w}y\# = uxv$, it holds that

$$(q, \hat{w}, \hat{w}y\#)\ {}_{p}^{a}\!\vdash_{\mathrm{M}}^{|\hat{w}|} (q, \varepsilon, y\#)$$

which completes the induction step. □

Consider any $w \in \mathrm{L(G)}$. Observe that $w = w'\flat$, where $w' \in \Sigma^*$. Next, observe that G generates w as

$$\begin{array}{llll}
(\mathrm{Z}, \bar{s}) & \Rightarrow_{\mathrm{G}} & (\mathrm{S}\#, s) & [(\mathrm{Z}, \bar{s}) \rightarrow (\mathrm{S}\#, s)] \\
 & \Rightarrow_{\mathrm{G}}^{*} & (w'\#, q) & (\text{Claim 5.1.7 with } y = \varepsilon) \\
 & \Rightarrow_{\mathrm{G}} & (w'\flat, \bar{f}) & [(\#, q) \rightarrow (\flat, \bar{f})]
\end{array}$$

where $q \in \mathrm{Q}$, and $(\mathrm{Z}, \bar{s}) \to (\mathrm{S\#}, s) \in \mathrm{P}$ and $(\#, q) \to (\flat, \bar{f}) \in \mathrm{P}$ are rules introduced in steps (i) and (iii) of the construction of P, respectively. Thus, $(s, w', \mathrm{S\#})\ {}^{a}\!\vdash^{*}_{\mathrm{M}} (q, \varepsilon, \#)$. Thus, $w'\flat \in \mathrm{L(G)}$ implies $w' \in \mathrm{L(M)}$, so $\mathrm{L(G)} \subseteq \mathrm{L(M)}\{\flat\}$.

Claim 5.1.8. *Let* $(p, w, \mathrm{S\#})\ {}^{a}\!\vdash^{m}_{\mathrm{M}} (q, \varepsilon, \hat{w}y\#)$*, where* $p, q \in \mathrm{Q}$*,* $w, \hat{w} \in \Sigma^*$*,* $y \in ((\mathrm{N} - \{\#\})\Sigma^*)^*$*, and* $m \geq 0$*. Then,* $(\mathrm{S\#}, p) \Rightarrow^{*}_{\mathrm{G}} (w\hat{w}y\#, q)$*.*

Proof. This claim is proved by induction on $m \geq 0$.

Basis. Let $m = 0$. Then, $w = \hat{w} = \varepsilon$, $y = \mathrm{S}$, and $(p, \varepsilon, \mathrm{S\#})\ {}^{a}\!\vdash^{0}_{\mathrm{M}} (p, \varepsilon, \mathrm{S\#})$. As $(\mathrm{S\#}, p) \Rightarrow^{0}_{\mathrm{G}} (\mathrm{S\#}, p)$, the basis holds.

Induction Hypothesis. Assume that the claim holds for all $0 \leq m \leq k$, where k is a non-negative integer.

Induction Step. Let $(p, w, \mathrm{S\#})\ {}^{a}\!\vdash^{k+1}_{\mathrm{M}} (q, \varepsilon, \hat{w}y\#)$, where $p, q \in \mathrm{Q}$, $w, \hat{w} \in \Sigma^*$, and $y \in ((\mathrm{N} - \{\#\})\Sigma^*)^*$. Since $k + 1 \geq 1$, express $(p, w, \mathrm{S\#})\ {}^{a}\!\vdash^{k+1}_{\mathrm{M}} (q, \varepsilon, \hat{w}y\#)$ as $(p, w, \mathrm{S\#})\ {}^{a}\!\vdash^{k}_{\mathrm{M}} \chi\ {}^{a}\!\vdash_{\mathrm{M}} (q, \varepsilon, \hat{w}y\#)$, where χ is a configuration of M whose form depends on whether the last move is a popping move or an expansion.

(I) Assume that $\chi\ {}^{a}_{p}\!\vdash_{\mathrm{M}} (q, \varepsilon, \hat{w}y\#)$. More specifically, let

$$\chi = (q, a, a\hat{w}y\#)$$

with $a \in \Sigma$ such that $w = w'a$, where $w' \in \Sigma^*$. Thus

$$(p, w, \mathrm{S\#})\ {}^{a}\!\vdash^{k}_{\mathrm{M}} (q, a, a\hat{w}y\#)\ {}^{a}_{p}\!\vdash_{\mathrm{M}} (q, \varepsilon, \hat{w}y\#)$$

Since $(p, w, \mathrm{S\#})\ {}^{a}\!\vdash^{k}_{\mathrm{M}} (q, a, a\hat{w}y\#)$, it holds that $(p, w', \mathrm{S\#})\ {}^{a}\!\vdash^{k}_{\mathrm{M}} (q, \varepsilon, a\hat{w}y\#)$. By the induction hypothesis, $(\mathrm{S\#}, p) \Rightarrow^{*}_{\mathrm{G}} (w'a\hat{w}y\#, q)$. As $w = w'a$, $(\mathrm{S\#}, p) \Rightarrow^{*}_{\mathrm{G}} (w\hat{w}y\#, q)$.

(II) Assume that $\chi\ {}^{a}_{e}\!\vdash_{\mathrm{M}} (q, \varepsilon, \hat{w}y\#)$. More specifically, let $\chi = (t, \varepsilon, u\mathrm{A}v)$, where $t \in \mathrm{Q}$, $\mathrm{A} \in \mathrm{N}$, $u \in ((\mathrm{N} - \{\#\})\Sigma^*)^*$, and $v \in \Gamma^*$. By the induction hypothesis, $(p, w, \mathrm{S\#})\ {}^{a}\!\vdash^{k}_{\mathrm{M}} (t, \varepsilon, u\mathrm{A}v)$ implies $(\mathrm{S\#}, p) \Rightarrow^{*}_{\mathrm{G}} (wu\mathrm{A}v, t)$. Consider that $(t, \varepsilon, u\mathrm{A}v)\ {}^{a}_{e}\!\vdash_{\mathrm{M}} (q, \varepsilon, uxv)\ [t\mathrm{A} \to qx]$, $t\mathrm{A} \to qx \in \mathrm{R}$, with $\hat{w}y\# = uxv$. Following the construction of P, there must be a rule $(\mathrm{A}, t) \to (x, q) \in \mathrm{P}$ introduced in step (ii). As there is no other rule $r_{\mathrm{P}} \in \mathrm{P}$ such that $\mathrm{lhs}(r_{\mathrm{P}}) = (\mathrm{A}', t)$, there is also no other rule $r_{\mathrm{R}} \in \mathrm{R}$ such that $\mathrm{lhs}(r_{\mathrm{R}}) = t\mathrm{A}'$, for all $\mathrm{A}' \in (\mathrm{alph}(u) - \Sigma)$. Thus, $(wu\mathrm{A}v, t) \Rightarrow_{\mathrm{G}} (wuxv, q)$, and by putting the previous sequences of derivations together, $(\mathrm{S\#}, p) \Rightarrow^{*}_{\mathrm{G}} (w\hat{w}y\#, q)$ since $\hat{w}y\# = uxv$. □

By the previous claim for $y = \hat{w} = \varepsilon$, if $(s, w, \mathrm{S\#})\ {}^{a}\!\vdash^{*}_{\mathrm{M}} (q, \varepsilon, \#)$, where $q \in \mathrm{Q}$ and $w \in \Sigma^*$, then $(\mathrm{S\#}, s) \Rightarrow^{*}_{\mathrm{G}} (w\#, q)$. As P contains rules introduced in steps (i) and (iii), it also holds that $(\mathrm{Z}, \bar{s}) \Rightarrow_{\mathrm{G}} (\mathrm{S\#}, s) \Rightarrow^{*}_{\mathrm{G}} (w\#, q) \Rightarrow_{\mathrm{G}} (w\flat, \bar{f})$. Thus, $w \in \mathrm{L(M)}$ implies $w\flat \in \mathrm{L(G)}$, so $\mathrm{L(M)}\{\flat\} \subseteq \mathrm{L(G)}$.

As $L(M)\{\flat\} \subseteq L(G)$ and $L(G) \subseteq L(M)\{\flat\}$, $L(M)\{\flat\} = L(G)$. Thus, Lemma 5.1.6 holds. Since the construction of G from M preserves the propagating property, the lemma also holds for propagating absolutely unlimited deep pushdown automata and state grammars. □

Theorem 5.1.9. AUDPDA = RE

Proof. Since **ST** = **RE**, **RE** ⊆ **AUDPDA** follows directly from Lemma 5.1.3. The rest directly follows from Church's Thesis. □

Theorem 5.1.10. AUDPDA$^{-\varepsilon}$ = CS

Proof. Inclusion **CS** ⊆ **AUDPDA**$^{-\varepsilon}$ follows directly from Lemma 5.1.3.

From Lemma 5.1.6, it follows that for every propagating AUDPDA M, it holds $L(M)\{\flat\} \in$ **CS**, where $\flat \notin \bigcup_{x \in L(M)} \mathrm{alph}(x)$. Since **CS** is closed under linear erasing, it also holds that $L(M) \in$ **CS** and hence **AUDPDA**$^{-\varepsilon}$ ⊆ **CS**. □

5.2 Relatively Unlimited Deep Pushdown Automata

The concept of expansion relative to the depth on pushdown was first introduced by Meduna and Křivka in [50] in the form of *relatively deep top-down parsers*, but they were never studied. In a relatively deep top-down parser, an expansion is done in three ways: let X be an expandable pushdown symbol at a given position. Then, either (1) expand the next pushdown symbol that is above X and move the position further down to the next expandable pushdown symbol, (2) expand the next pushdown symbol that is below X and move the position up to the next expandable pushdown symbol, or (3) just expand X and set the position to the next expandable pushdown symbol with the same depth as X.

Relatively unlimited deep pushdown automata originate from the relatively deep top-down parsers, but their definition of expansion is adjusted to be more intuitive: they first expand X and then they move the position up, down, or stay still.

Definition 5.2.1. A *relatively unlimited deep pushdown automaton* (abbreviated RUDPDA), M, is an 8-tuple $M = (Q, \Sigma, \Gamma, \#, R, s, S, F)$, where

- $Q, \Sigma, \Gamma, \#, s, S, F$ are defined as in Definition 5.1.1;
- $R \subseteq (\{-1, 0, 1\} \times Q \times (\Gamma - (\Sigma \cup \{\#\})) \times Q \times (\Gamma - \{\#\})^*) \cup (Q \times \{\#\} \times Q \times (\Gamma - \{\#\})^* \{\#\})$ is a finite set of *rules*.

Rules $(k, p, A, q, x) \in R$ and $(p, A, q, x) \in R$ are usually written as $kpA \rightarrow qx \in R$ and $pA \rightarrow qx \in R$, respectively. M is said to be *propagating* if $kpA \rightarrow qx \in R$ implies $x \neq \varepsilon$.

Set $\Xi = Q \times \Sigma^* \times (\Gamma - \{\#\})^*\{\#\} \times \mathbb{N}$. Then, $\chi \in \Xi$ is said to be a *configuration* of M.

The M-based relation of *direct pop move*, ${}_{p}^{r}\vdash_{M}$, over Ξ is defined as follows:

$$(p, aw, az, i)\ {}_{p}^{r}\vdash_{M} (p, w, z, i)$$

where $p \in \mathrm{Q}$, $a \in \Sigma$, $w \in \Sigma^*$, $z \in \Gamma^*$, and $i \in \mathbb{N}$.

The M-based relation of *direct expansion move*, ${}_{e}^{r}\vdash_{\mathrm{M}}$, over Ξ is defined as follows:

$$(p, w, u\mathrm{A}v, i) \; {}_{e}^{r}\vdash_{\mathrm{M}} \; (q, w, uxv, j)$$

if and only if $kp\mathrm{A} \rightarrow qx \in \mathrm{R}$, where $p, q \in \mathrm{Q}$, $w \in \Sigma^*$, $u, x, v \in \Gamma^*$, $i, j \in \mathbb{N}$, and

- either $\mathrm{A} \in \Gamma - (\Sigma \cup \{\#\})$, $k \in \{-1, 0, 1\}$, and $j = i + k = \mathscr{O}_{\Gamma-\Sigma}(u\mathrm{A}) + k$;
- or $\mathrm{A} = \#$, $k = \varepsilon$, and $j = i = \mathscr{O}_{\Gamma-\Sigma}(u\mathrm{A})$.

The M-based relation of *direct move*, ${}^{r}\vdash_{\mathrm{M}}$, is then defined as ${}^{r}\vdash_{\mathrm{M}} = {}_{p}^{r}\vdash_{\mathrm{M}} \cup {}_{e}^{r}\vdash_{\mathrm{M}}$. For $m \geq 0$, ${}_{p}^{r}\vdash_{\mathrm{M}}^{m}$, ${}_{e}^{r}\vdash_{\mathrm{M}}^{m}$, ${}^{r}\vdash_{\mathrm{M}}^{m}$, ${}_{p}^{r}\vdash_{\mathrm{M}}^{+}$, ${}_{e}^{r}\vdash_{\mathrm{M}}^{+}$, ${}^{r}\vdash_{\mathrm{M}}^{+}$, ${}_{p}^{r}\vdash_{\mathrm{M}}^{*}$, ${}_{e}^{r}\vdash_{\mathrm{M}}^{*}$, and ${}^{r}\vdash_{\mathrm{M}}^{*}$ are defined as usual.

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

$$\mathrm{L(M)} = \{w \mid (s, w, \mathrm{S}\#, 1) \; {}^{r}\vdash_{\mathrm{M}}^{*} \; (f, \varepsilon, \#, 1), w \in \Sigma^*, f \in \mathrm{F}\}$$

Additionally, the language accepted by M by empty pushdown, $\mathrm{L(M)}_{\varepsilon}$, is defined as

$$\mathrm{L(M)}_{\varepsilon} = \{w \mid (s, w, \mathrm{S}\#, 1) \; {}^{r}\vdash_{\mathrm{M}}^{*} \; (q, \varepsilon, \#, 1), w \in \Sigma^*, q \in \mathrm{Q}\}$$

RUDPDA and **RUDPDA**$^{-\varepsilon}$ denote the families of languages accepted by RUDPDA and propagating RUDPDA, respectively. □

In the following example, it is demonstrated how relatively deep pushdown automata work during a process of accepting a word from a non-context-free language.

Example 5.2.2. Consider the relatively unlimited deep pushdown automaton

$$\mathrm{M} = (\mathrm{Q}, \{a, b, c\}, \{\mathrm{S}, \mathrm{A}, \mathrm{B}, \mathrm{C}, \blacktriangle, \blacksquare, \#, a, b, c\}, \#, \mathrm{R}, \langle s \rangle, \mathrm{S}, \{\langle f \rangle\})$$

where Q is induced by R and R contains rules

$$\begin{array}{rcl@{\qquad\qquad}rcl}
0\langle s\rangle\mathrm{S} & \rightarrow & \langle ra\rangle a\mathrm{A}\blacksquare & 1\langle x\mathrm{B}\rangle\mathrm{C} & \rightarrow & \langle x\mathrm{B}\rangle\mathrm{C} \\
0\langle ra\rangle\mathrm{A} & \rightarrow & \langle ra\rangle a\mathrm{AA} & 0\langle x\mathrm{B}\rangle\mathrm{B} & \rightarrow & \langle x\mathrm{A}\rangle \\
0\langle ra\rangle\mathrm{A} & \rightarrow & \langle rb\rangle b\mathrm{BA} & 1\langle x\mathrm{A}\rangle\mathrm{B} & \rightarrow & \langle x\mathrm{A}\rangle\mathrm{B} \\
0\langle rb\rangle\mathrm{B} & \rightarrow & \langle rb\rangle b\mathrm{BB} & 0\langle x\mathrm{A}\rangle\mathrm{A} & \rightarrow & \langle r\rangle \\
0\langle rb\rangle\mathrm{B} & \rightarrow & \langle rc\rangle c\mathrm{CB} & (-1)\langle r\rangle\blacksquare & \rightarrow & \langle r\rangle\blacksquare \\
0\langle rc\rangle\mathrm{C} & \rightarrow & \langle rc\rangle c\mathrm{CC} & (-1)\langle r\rangle\mathrm{A} & \rightarrow & \langle r\rangle\mathrm{A} \\
0\langle rc\rangle\mathrm{C} & \rightarrow & \langle t\rangle\blacktriangle\mathrm{C} & (-1)\langle r\rangle\mathrm{B} & \rightarrow & \langle r\rangle\mathrm{B} \\
1\langle t\rangle\blacktriangle & \rightarrow & \langle x\mathrm{C}\rangle\blacktriangle & (-1)\langle r\rangle\mathrm{C} & \rightarrow & \langle r\rangle\mathrm{C} \\
(-1)\langle x\mathrm{C}\rangle\blacksquare & \rightarrow & \langle h\rangle & 0\langle r\rangle\blacktriangle & \rightarrow & \langle t\rangle\blacktriangle \\
0\langle x\mathrm{C}\rangle\mathrm{C} & \rightarrow & \langle x\mathrm{B}\rangle & 0\langle h\rangle\blacktriangle & \rightarrow & \langle f\rangle
\end{array}$$

Then a word *aabbcc* is accepted by M in the following way:

$$
\begin{array}{lll}
(\langle s\rangle, aabbcc, \mathrm{S}\#, 1) & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle ra\rangle, aabbcc, a\mathrm{A}\blacksquare\#, 1) & [0\langle s\rangle \mathrm{S} \rightarrow \langle ra\rangle a\mathrm{A}\blacksquare] \\
 & {}_{p}^{r}\!\vdash_{\mathrm{M}} (\langle ra\rangle, abbcc, \mathrm{A}\blacksquare\#, 1) & \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle ra\rangle, abbcc, a\mathrm{AA}\blacksquare\#, 1) & [0\langle ra\rangle \mathrm{A} \rightarrow \langle ra\rangle a\mathrm{AA}] \\
 & {}_{p}^{r}\!\vdash_{\mathrm{M}} (\langle ra\rangle, bbcc, \mathrm{AA}\blacksquare\#, 1) & \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle rb\rangle, bbcc, b\mathrm{BAA}\blacksquare\#, 1) & [0\langle ra\rangle \mathrm{A} \rightarrow \langle rb\rangle b\mathrm{BA}] \\
 & {}_{p}^{r}\!\vdash_{\mathrm{M}} (\langle rb\rangle, bcc, \mathrm{BAA}\blacksquare\#, 1) & \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle rb\rangle, bcc, b\mathrm{BBAA}\blacksquare\#, 1) & [0\langle rb\rangle \mathrm{B} \rightarrow \langle rb\rangle b\mathrm{BB}] \\
 & {}_{p}^{r}\!\vdash_{\mathrm{M}} (\langle rb\rangle, cc, \mathrm{BBAA}\blacksquare\#, 1) & \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle rc\rangle, cc, c\mathrm{CBBAA}\blacksquare\#, 1) & [0\langle rb\rangle \mathrm{B} \rightarrow \langle rc\rangle c\mathrm{CB}] \\
 & {}_{p}^{r}\!\vdash_{\mathrm{M}} (\langle rc\rangle, c, \mathrm{CBBAA}\blacksquare\#, 1) & \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle rc\rangle, c, c\mathrm{CCBBAA}\blacksquare\#, 1) & [0\langle rc\rangle \mathrm{C} \rightarrow \langle rc\rangle c\mathrm{CC}] \\
 & {}_{p}^{r}\!\vdash_{\mathrm{M}} (\langle rc\rangle, \varepsilon, \mathrm{CCBBAA}\blacksquare\#, 1) & \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle t\rangle, \varepsilon, \blacktriangle\mathrm{CCBBAA}\blacksquare\#, 1) & [0\langle rc\rangle \mathrm{C} \rightarrow \langle t\rangle \blacktriangle\mathrm{C}] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle x\mathrm{C}\rangle, \varepsilon, \blacktriangle\mathrm{CCBBAA}\blacksquare\#, 2) & [1\langle t\rangle \blacktriangle \rightarrow \langle x\mathrm{C}\rangle \blacktriangle] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle x\mathrm{B}\rangle, \varepsilon, \blacktriangle\mathrm{CBBAA}\blacksquare\#, 2) & [0\langle x\mathrm{C}\rangle \mathrm{C} \rightarrow \langle x\mathrm{B}\rangle] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle x\mathrm{B}\rangle, \varepsilon, \blacktriangle\mathrm{CBBAA}\blacksquare\#, 3) & [1\langle x\mathrm{B}\rangle \mathrm{C} \rightarrow \langle x\mathrm{B}\rangle \mathrm{C}] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle x\mathrm{A}\rangle, \varepsilon, \blacktriangle\mathrm{CBAA}\blacksquare\#, 3) & [0\langle x\mathrm{B}\rangle \mathrm{B} \rightarrow \langle x\mathrm{A}\rangle] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle x\mathrm{A}\rangle, \varepsilon, \blacktriangle\mathrm{CBAA}\blacksquare\#, 4) & [1\langle x\mathrm{A}\rangle \mathrm{B} \rightarrow \langle x\mathrm{A}\rangle \mathrm{B}] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle r\rangle, \varepsilon, \blacktriangle\mathrm{CBA}\blacksquare\#, 4) & [0\langle x\mathrm{A}\rangle \mathrm{A} \rightarrow \langle r\rangle] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle r\rangle, \varepsilon, \blacktriangle\mathrm{CBA}\blacksquare\#, 3) & [(-1)\langle r\rangle \mathrm{A} \rightarrow \langle r\rangle \mathrm{A}] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle r\rangle, \varepsilon, \blacktriangle\mathrm{CBA}\blacksquare\#, 2) & [(-1)\langle r\rangle \mathrm{B} \rightarrow \langle r\rangle \mathrm{B}] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle r\rangle, \varepsilon, \blacktriangle\mathrm{CBA}\blacksquare\#, 1) & [(-1)\langle r\rangle \mathrm{C} \rightarrow \langle r\rangle \mathrm{C}] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle t\rangle, \varepsilon, \blacktriangle\mathrm{CBA}\blacksquare\#, 1) & [0\langle r\rangle \blacktriangle \rightarrow \langle t\rangle \blacktriangle] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle x\mathrm{C}\rangle, \varepsilon, \blacktriangle\mathrm{CBA}\blacksquare\#, 2) & [1\langle t\rangle \blacktriangle \rightarrow \langle x\mathrm{C}\rangle \blacktriangle] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle x\mathrm{B}\rangle, \varepsilon, \blacktriangle\mathrm{BA}\blacksquare\#, 2) & [0\langle x\mathrm{C}\rangle \mathrm{C} \rightarrow \langle x\mathrm{B}\rangle] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle x\mathrm{A}\rangle, \varepsilon, \blacktriangle\mathrm{A}\blacksquare\#, 2) & [0\langle x\mathrm{B}\rangle \mathrm{B} \rightarrow \langle x\mathrm{A}\rangle] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle r\rangle, \varepsilon, \blacktriangle\blacksquare\#, 2) & [0\langle x\mathrm{A}\rangle \mathrm{A} \rightarrow \langle r\rangle] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle r\rangle, \varepsilon, \blacktriangle\blacksquare\#, 1) & [(-1)\langle r\rangle \blacksquare \rightarrow \langle r\rangle \blacksquare] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle t\rangle, \varepsilon, \blacktriangle\blacksquare\#, 1) & [0\langle r\rangle \blacktriangle \rightarrow \langle t\rangle \blacktriangle] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle x\mathrm{C}\rangle, \varepsilon, \blacktriangle\blacksquare\#, 2) & [1\langle t\rangle \blacktriangle \rightarrow \langle x\mathrm{C}\rangle \blacktriangle] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle h\rangle, \varepsilon, \blacktriangle\#, 1) & [(-1)\langle x\mathrm{C}\rangle \blacksquare \rightarrow \langle h\rangle] \\
 & {}_{e}^{r}\!\vdash_{\mathrm{M}} (\langle f\rangle, \varepsilon, \#, 1) & [0\langle h\rangle \blacktriangle \rightarrow \langle f\rangle]
\end{array}
$$

In short, $(\langle s\rangle, aabbcc, \mathrm{S}\#, 1)\ {}^{r}\!\vdash_{\mathrm{M}}^{*} (\langle f\rangle, \varepsilon, \#, 1)$.Observe that $\mathrm{L(M)} = \mathrm{L(M)}_{\varepsilon} = \{a^n b^n c^n \mid n \geq 1\}$, which belongs to **CS** − **CF**. □

Accepting Power

Following theorems show that also relatively unlimited deep pushdown automata and their propagating variants can accept every recursively enumerable and context-sensitive language, respectively.

Theorem 5.2.3. RUDPDA = RE

Proof. **RUDPDA** $\subseteq$ **RE** follows directly from Church's Thesis. To demonstrate that **RE** $\subseteq$ **RUDPDA**, let $L \in$ **RE**. Without any loss on generality, suppose that there exists a Turing machine $M = (Q, \Sigma, \Gamma, \Delta, R, s, F)$ such that

$$L = L(M) = \{w \mid sw \vdash_M^* fw\alpha, w \in \Sigma^*, \alpha \in (\Gamma - \Sigma)^*, f \in F\}$$

From M, construct a RUDPDA $M' = (Q', \Sigma, \Gamma', \#', R', s', S', \{f'\})$, $(Q \cup \Gamma) \cap (Q' \cup \Gamma') = \Sigma$, such that $L(M') = L(M)$, in the following way:

1. Set $Q' = \{s', f'\} \cup \{\langle q \rangle \mid q \in Q\} \cup \{\langle r \rangle \mid r \in R\}$.
2. Set $\Gamma' = \{S', Z', \#'\} \cup \{\bar{X} \mid X \in \Gamma\} \cup \Sigma$.
3. Set $R'_0 = \{0s'S' \rightarrow s'S'\bar{a}, 0s'S' \rightarrow \langle s \rangle \bar{a} \mid a \in \Sigma\}$.
4. Set $R'_1 = \{1\langle p \rangle \bar{X} \rightarrow \langle q \rangle \bar{Y} \mid pX \rightarrow qYd_R \in R\}$.
5. Set $R'_2 = \{\langle p \rangle \#' \rightarrow \langle r \rangle Z'\#', 1\langle r \rangle Z' \rightarrow \langle q \rangle \bar{Y} \mid r\colon p\Delta \rightarrow qYd_R \in R\}$.
6. Set $R'_3 = \{(-1)\langle p \rangle \bar{X} \rightarrow \langle q \rangle \bar{Y} \mid pX \rightarrow qYd_L \in R\}$.
7. Set $R'_4 = \{\langle p \rangle \#' \rightarrow \langle r \rangle Z'\#', (-1)\langle r \rangle Z' \rightarrow \langle q \rangle \bar{Y} \mid r\colon p\Delta \rightarrow qYd_L \in R\}$.
8. Set $R'_5 = \{0\langle f \rangle \bar{X} \rightarrow f'\bar{X} \mid X \in \Gamma\} \cup \{0f'\bar{a} \rightarrow f'a \mid a \in \Sigma\} \cup \{0f'\bar{X} \rightarrow f' \mid X \in (\Gamma - \Sigma)\}$.
9. Set $R' = \bigcup_{0 \leq i \leq 5} R'_i$.

First, M′ applies rules from R'_0 to prepare M's tape on the pushdown. Then, by applying rules from $\bigcup_{1 \leq i \leq 4} R'_i$, M′ simulates moves performed by M. Finally, by applying rules from R'_5, M′ empties the pushdown and finishes its computation.

Therefore,

$$\begin{aligned}(s', w, S'\#', 1) &\ {}^r\!\vdash_{M'}^* (\langle s \rangle, w, h(w)\#', 1)\\ &\ {}^r\!\vdash_{M'}^* (\langle f \rangle, w, h(w\alpha)\#', 1)\\ &\ {}^r\!\vdash_{M'} (f', w, h(w\alpha)\#', 1)\\ &\ {}^r\!\vdash_{M'}^* (f', \varepsilon, \#', 1)\end{aligned}$$

if and only if $sw \vdash_M^* fw\alpha$, where h is a morphism from Γ^* to Γ'^* defined as $h(a) = \bar{a}$. □

Theorem 5.2.4. RUDPDA$^{-\varepsilon}$ = CS

Proof. First, it will be demonstrated that for every propagating RUDPDA

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

there exists a linear bounded automaton $M' = (Q', \Sigma, \Gamma', \Delta', R', s', \{f'\})$ that simulates M. Let $w \in L(M)$. Then, M′ will accept w by performing the following steps:

(I) First, M′ rewrites its initial configuration $s'w$ to the configuration that represents the initial configuration of M. Observe that as M is propagating, the number of symbols on its pushdown does not exceed $|w| + 1$. Therefore, every input symbol is replaced by M' by a symbol of the form $\langle a, \mathrm{X}, \mathrm{Y}\rangle$, where $a \in \Sigma$ is the original input symbol, $\mathrm{X} \in (\Gamma \cup \{\square\})$ is a pushdown symbol, $\square \notin \Gamma$ denotes a free space on a pushdown, and $\mathrm{Y} \in (\{+, -\} \cup \Omega)$ is an auxiliary symbol, where $+$ denotes that input symbol has not been read by M yet, $-$ denotes that input symbol has been read by M and simultaneously popped from M's pushdown, and Ω is an alphabet of symbols used to temporarily mark the current position of M''s head on the tape. Note that from the M''s point of view, $-$ works as a left bounder, and M' should not move its head left behind $-$. Also note that the bottom symbol, #, is not encoded directly, but is implicit. Now, M' contains three tapes in one—input, pushdown, and auxiliary—without exceeding the space given by w.

Summing it up, if $w = a_1a_2a_3 \dots a_n$, then

$$s'a_1a_2a_3 \dots a_n \vdash^*_{\mathrm{M}'} \langle s\rangle \begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ \mathrm{S} & \square & \square & \dots & \square \\ + & + & + & \dots & + \end{bmatrix}$$

in this step, where $\langle s\rangle \in \mathrm{Q}'$ is a state that corresponds s.

(II) Suppose that

$$(p, w', u\mathrm{A}v, i) \; {}^{r}_{e}\!\vdash_{\mathrm{M}} (q, w', uxv, i + 1)\, [1p\mathrm{A} \to qx]$$

where $p, q \in \mathrm{Q}$, $w' \in \Sigma^*$, $\mathrm{A} \in (\Gamma - (\Sigma \cup \{\#\}))$, $u, v, x \in \Gamma^*$, $i \in \mathbb{N}$, $1p\mathrm{A} \to qx \in \mathrm{R}$, is going to be simulated by M'. Then, M' takes the following actions:

1. M' ensures there is enough space to rewrite A with x on the pushdown tape. If there is not enough space, then it means that the size of the M's pushdown exceeds $|w|$ and thus M never accepts its input.
2. M' moves v $(|x| - 1)$ cells to the right.
3. M' replaces A with x.
4. If $xv = x_1\mathrm{X}_1x_2\mathrm{X}_2v_2$, $x_1, x_2 \in \Sigma^*$, $\mathrm{X}_1, \mathrm{X}_2 \in (\Gamma - (\Sigma \cup \{\#\}))$, $v_2 \in \Gamma^*$, M' moves its head to X_2. Otherwise, M' moves its head to the end of pushdown tape and records the information about missing symbols inside its state. This reflects the situation that M, to make accepting its input possible, must apply $q\# \to ty\#$, $t \in \mathrm{Q}$, $y \in (\Gamma - \{\#\})^*$ as its next rule.
5. M' enters the state corresponding to q.

(III) Suppose that

$$(p, w', u\mathrm{A}v, i) \; {}^{r}_{e}\!\vdash_{\mathrm{M}} (q, w', uxv, i - 1)\, [(-1)p\mathrm{A} \to qx]$$

where $p, q \in \mathrm{Q}$, $w' \in \Sigma^*$, $\mathrm{A} \in (\Gamma - (\Sigma \cup \{\#\}))$, $u, v, x \in \Gamma^*$, $i \in \mathbb{N}$, $(-1)p\mathrm{A} \to qx \in \mathrm{R}$, is going to be simulated by M'. Then, M' takes the following actions:

1. M' replaces A with x by following the steps (II.1) to (II.3).

2. If $u = u_1 X_1 u_2, u_1 \in \Gamma^*, X_1 \in (\Gamma - (\Sigma \cup \{\#\})), u_2 \in \Sigma^*$, M' moves its head to X_1. Otherwise, M'—and by extension M—will be blocked and unable to accept w.
3. M' enters the state corresponding to q.

(IV) Suppose that

$$(p, w', uAv, i) \, {}_{e}^{r}\vdash_M (q, w', uxv, i) \, [0pA \to qx]$$

where $p, q \in Q$, $w' \in \Sigma^*$, $A \in (\Gamma - (\Sigma \cup \{\#\}))$, $u, v, x \in \Gamma^*$, $i \in \mathbb{N}$, $0pA \to qx \in R$, is to be simulated by M'. Then, M' takes the following actions:

1. M' replaces A with x by following the steps (II.1) to (II.3).
2. If $xv = x_1 X_1 v_1, x_1 \in \Sigma^*, X_1 \in (\Gamma - (\Sigma \cup \{\#\})), v_1 \in \Gamma^*$, M' moves its head to X_1. Otherwise, like in step (II), M' moves its head to the end of pushdown tape and records the information about the missing symbol inside its state.
3. M' enters the state corresponding to q.

(V) Suppose that

$$(p, w', u\#, i) \, {}_{e}^{r}\vdash_M (q, w', ux\#, i) \, [p\# \to qx\#]$$

where $p, q \in Q$, $w' \in \Sigma^*$, $u, x \in (\Gamma - \{\#\})^*$, $i \in \mathbb{N}$, $p\# \to qx\# \in R$, is going to be simulated by M'. Then, M' takes the following actions:

1. If M' has not recorded that there are any missing symbols (on its pushdown tape) in its state, it saves its head's current position, records this information inside its state and moves its head to the end of pushdown tape.
2. M' ensures that there is enough space to append x to the end of pushdown tape and then performs the action.
3. If $x = x_1 X_1 x_2 X_2 y, x_1, x_2 \in \Sigma^*, X_1, X_2 \in (\Gamma - \Sigma), y \in \Gamma^*$ and in its state, M' has recorded that two symbols from $(\Gamma - (\Sigma \cup \{\#\}))$ are missing on its pushdown tape, then M' moves its head to X_2 and removes the recorded information from its state. The process is similar in case one symbol is missing. If $\mathscr{O}_{\Gamma-\Sigma}(x) = 1$ and two symbols are needed, M' updates the information in its state accordingly.
4. If M' has recorded that its head should be restored to its original position in its state, then M' restores it and removes the recorded information from its state.
5. M' enters the state corresponding to q.

(VI) Suppose that

$$(p, aw', az, i) \, {}_{p}^{r}\vdash_M (p, w', z, i)$$

where $p \in Q$, $a \in \Sigma$, $w' \in \Sigma^*$, $z \in \Gamma^*$, $i \in \mathbb{N}$, is going to be simulated by M'. Then, M' takes the following actions:

1. M' saves its head's current position.
2. M' finds the leftmost symbol $\langle a, a, + \rangle$ and rewrites it to the symbol $\langle a, a, - \rangle$.

3. M$'$ restores the original position of its head.

(VII) Suppose that M has accepted its input. That is, its configuration is $(f, \varepsilon, \#, 1)$, $f \in \mathrm{F}$. At this point, M$'$ finishes M's simulation and its configuration is

$$\begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ a_1 & a_2 & a_3 & \dots & a_n \\ - & - & - & \dots & - \end{bmatrix} \langle f \rangle$$

where $\langle f \rangle \in \mathrm{Q}'$ corresponds to f. Then,

$$\begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ a_1 & a_2 & a_3 & \dots & a_n \\ - & - & - & \dots & - \end{bmatrix} \langle f \rangle \vdash^*_{\mathrm{M}'} \begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ a_1 & a_2 & a_3 & \dots & a_n \\ - & - & - & \dots & - \end{bmatrix} f'$$

and thus w is accepted by M$'$ if and only if it is accepted by M.

To show that **CS** $\subseteq$ **RUDPDA**$^{-\varepsilon}$, suppose without any loss on generality that for every $\mathrm{L} \in$ **CS** there exists a linear bounded automaton $\mathrm{M} = (\mathrm{Q}, \Sigma, \Gamma, \Delta, \mathrm{R}, s, \mathrm{F})$ such that

$$\mathrm{L} = \mathrm{L}(\mathrm{M}) = \{w \mid sw \vdash^*_{\mathrm{M}} fw, w \in \Sigma^*, f \in \mathrm{F}\}$$

From M, construct a RUDPDA $\mathrm{M}' = (\mathrm{Q}', \Sigma, \Gamma', \#', \mathrm{R}', s', \mathrm{S}', \{f'\})$ by following the steps in Theorem 5.2.3. Observe that

$$(\langle s \rangle, w, h(w)\#', 1) \; {}^r\!\vdash^*_{\mathrm{M}'} (\langle f \rangle, w, h(w)\#', 1) \quad \text{if and only if} \quad sw \vdash^*_{\mathrm{M}} fw$$

and hence R'_5 can be constructed as

$$\mathrm{R}'_5 = \{0\langle f \rangle \bar{a} \to f'\bar{a}, 0f'\bar{a} \to f'\bar{a} \mid a \in \Sigma\}$$

Thus $(s', w, \mathrm{S}'\#', 1) \; {}^r\!\vdash^*_{\mathrm{M}'} (f', \varepsilon, \#', 1)$ if and only if $sw \vdash^*_{\mathrm{M}} fw$ and since M$'$ is propagating, **CS** $\subseteq$ **RUDPDA**$^{-\varepsilon}$. □

Chapter 6
Jumping Pure Grammars

In pure grammars, as recalled in Definition 2.3.1, all symbols from the total alphabet are terminal symbols. As a consequence, given a pure grammar G, every sentential form produced by G is also a word that belongs to L(G).

This chapter presents pure grammars extended with derivation modes that allow jumping rewriting, both in a sequential and parallel way. The focus is laid mostly on pure grammars with context-free rules and on relations between families of languages they generate.

The contents of this chapter, published in [49], are organized into four sections. Section 6.1 defines the notion of jumping pure grammars and the corresponding language families. In greater detail, jumping pure grammars are defined as pure grammars that can work in four derivation modes: classical sequential and parallel mode and jumping sequential and parallel mode, where jumping sequential mode in fact consist of two other modes—left jumping and right jumping sequential modes. Section 6.2 gives a survey of several elementary properties of jumping pure grammars. Section 6.3 studies mutual relations between **SPCF**, **JSPCF**, **PPCF**, **JPPCF**, **CF**, and **CS** families. Finally, Section 6.4 gives remarks on propagating jumping pure grammars and jumping pure grammars with unary alphabets.

6.1 Definitions

First, define six derivation modes for pure grammars. Classical sequential ($_s\Rightarrow$) and classical jumping ($_{lj}\Rightarrow$, $_{rj}\Rightarrow$, $_j\Rightarrow$) derivation modes are defined analogously with jumping grammars. Parallel sequential derivation mode ($_p\Rightarrow$) is a generalization of direct derivation in 0L systems and parallel jumping derivation mode ($_{jp}\Rightarrow$) adds the element of jumping to it.

Definition 6.1.1. Let $\mathrm{G} = (\Sigma, \mathrm{P}, \sigma)$ be a pure grammar. The six G-based relations of direct derivations, $_s\Rightarrow_{\mathrm{G}}$, $_{lj}\Rightarrow_{\mathrm{G}}$, $_{rj}\Rightarrow_{\mathrm{G}}$, $_j\Rightarrow_{\mathrm{G}}$, $_p\Rightarrow_{\mathrm{G}}$, and $_{jp}\Rightarrow_{\mathrm{G}}$, over Σ^* are defined as follows:

(i) $_s\Rightarrow_{\mathrm{G}}$, $_{lj}\Rightarrow_{\mathrm{G}}$, $_{rj}\Rightarrow_{\mathrm{G}}$, and $_j\Rightarrow_{\mathrm{G}}$ are defined exactly as in Definition 3.1.11 with the difference that they are defined over Σ^*.

(ii) For $u, v \in \Sigma^*$ and some $n \geq 1$, $u \;_p\Rightarrow_{\mathrm{G}} v$ if and only if $u = x_1x_2 \dots x_n$, $v = y_1y_2 \dots y_n$, and $x_i \to y_i \in \mathrm{P}$, where $x_i, y_i \in \Sigma^*$, $1 \leq i \leq n$.

(iii) For $u, v \in \Sigma^*$ and some $n \geq 1$, $u \;{}_{jp}\!\Rightarrow_{\mathrm{G}}\; v$ if and only if $u = x_1 x_2 \ldots x_n$, $v = y_{p(1)} y_{p(2)} \ldots y_{p(n)}$, $p \in \mathrm{perm}(\{1, 2, \ldots, n\})$, and $x_i \to y_i \in \mathrm{P}$, where $x_i, y_i \in \Sigma^*$, $1 \leq i \leq n$.

Let $h \in \{s, lj, rj, j, p, jp\}$. For $k \geq 0$, ${}_{h}\!\Rightarrow^{k}_{\mathrm{G}}$, ${}_{h}\!\Rightarrow^{+}_{\mathrm{G}}$, and ${}_{h}\!\Rightarrow^{*}_{\mathrm{G}}$ are defined as usual. The language generated by G in h-mode, $\mathrm{L}(\mathrm{G}, {}_{h}\!\Rightarrow)$, is defined as

$$\mathrm{L}(\mathrm{G}, {}_{h}\!\Rightarrow) = \{w \mid \sigma \;{}_{h}\!\Rightarrow^{*}_{\mathrm{G}}\; w, w \in \Sigma^*\}$$

□

Generally speaking, if a pure grammar works under a derivation mode defined in Definition 6.1.1, then it is referred to as a *jumping pure grammar*. Note that sequential derivation modes are special cases of the jumping ones.

To demonstrate how using the jumping derivation modes can influence generated language, consider a pure grammar $\mathrm{G} = (\Sigma, \mathrm{P}, a)$, where $\Sigma = \{a, b, c, d\}$ and

$$\mathrm{P} = \{a \to abcd, a \to a, b \to b, c \to c, d \to d\}$$

Observe that $\mathrm{L}(\mathrm{G}, {}_{s}\!\Rightarrow) = \mathrm{L}(\mathrm{G}, {}_{p}\!\Rightarrow) = \{a\}\{bcd\}^*$ is a regular language, but

$$\mathrm{L}(\mathrm{G}, {}_{j}\!\Rightarrow) = \mathrm{L}(\mathrm{G}, {}_{jp}\!\Rightarrow) = \{w \mid \mathscr{O}_a(w) = 1, \mathscr{O}_b(w) = \mathscr{O}_c(w) = \mathscr{O}_d(w), w \in \Sigma^+\}$$

is a non-context-free language.

Next, families of languages generated by pure grammars in h-mode, where $h \in \{s, lj, rj, j, p, jp\}$, are defined.

Definition 6.1.2. Let $\mathscr{G}(\mathrm{PG})$, $\mathscr{G}(\mathrm{PG}^{-\varepsilon})$, $\mathscr{G}(\mathrm{PCFG})$, and $\mathscr{G}(\mathrm{PCFG}^{-\varepsilon})$ denote the set of all pure grammars, the set of all propagating pure grammars, the set of all pure context-free grammars, and the set of all propagating pure context-free grammars, respectively. Then

1. $\mathbf{SP} = \{\mathrm{L}(\mathrm{G}, {}_{s}\!\Rightarrow) \mid \mathrm{G} \in \mathscr{G}(\mathrm{PG})\}$
2. $\mathbf{SP}^{-\varepsilon} = \{\mathrm{L}(\mathrm{G}, {}_{s}\!\Rightarrow) \mid \mathrm{G} \in \mathscr{G}(\mathrm{PG}^{-\varepsilon})\}$
3. $\mathbf{JSP} = \{\mathrm{L}(\mathrm{G}, {}_{j}\!\Rightarrow) \mid \mathrm{G} \in \mathscr{G}(\mathrm{PG})\}$
4. $\mathbf{JSP}^{-\varepsilon} = \{\mathrm{L}(\mathrm{G}, {}_{j}\!\Rightarrow) \mid \mathrm{G} \in \mathscr{G}(\mathrm{PG}^{-\varepsilon})\}$
5. $\mathbf{PP} = \{\mathrm{L}(\mathrm{G}, {}_{p}\!\Rightarrow) \mid \mathrm{G} \in \mathscr{G}(\mathrm{PG})\}$
6. $\mathbf{PP}^{-\varepsilon} = \{\mathrm{L}(\mathrm{G}, {}_{p}\!\Rightarrow) \mid \mathrm{G} \in \mathscr{G}(\mathrm{PG}^{-\varepsilon})\}$
7. $\mathbf{JPP} = \{\mathrm{L}(\mathrm{G}, {}_{jp}\!\Rightarrow) \mid \mathrm{G} \in \mathscr{G}(\mathrm{PG})\}$
8. $\mathbf{JPP}^{-\varepsilon} = \{\mathrm{L}(\mathrm{G}, {}_{jp}\!\Rightarrow) \mid \mathrm{G} \in \mathscr{G}(\mathrm{PG}^{-\varepsilon})\}$
9. $\mathbf{SPCF} = \{\mathrm{L}(\mathrm{G}, {}_{s}\!\Rightarrow) \mid \mathrm{G} \in \mathscr{G}(\mathrm{PCFG})\}$
10. $\mathbf{SPCF}^{-\varepsilon} = \{\mathrm{L}(\mathrm{G}, {}_{s}\!\Rightarrow) \mid \mathrm{G} \in \mathscr{G}(\mathrm{PCFG}^{-\varepsilon})\}$

11. $\mathbf{JSPCF} = \{L(G, {}_j\Rightarrow) \mid G \in \mathscr{G}(PCFG)\}$
12. $\mathbf{JSPCF}^{-\varepsilon} = \{L(G, {}_j\Rightarrow) \mid G \in \mathscr{G}(PCFG^{-\varepsilon})\}$
13. $\mathbf{PPCF} = \{L(G, {}_p\Rightarrow) \mid G \in \mathscr{G}(PCFG)\}$
14. $\mathbf{PPCF}^{-\varepsilon} = \{L(G, {}_p\Rightarrow) \mid G \in \mathscr{G}(PCFG^{-\varepsilon})\}$
15. $\mathbf{JPPCF} = \{L(G, {}_{jp}\Rightarrow) \mid G \in \mathscr{G}(PCFG)\}$
16. $\mathbf{JPPCF}^{-\varepsilon} = \{L(G, {}_{jp}\Rightarrow) \mid G \in \mathscr{G}(PCFG^{-\varepsilon})\}$ □

6.2 Elementary Properties

This section introduces some properties of language families defined in Definition 6.1.2, which will be referred in proofs in the next section.

Theorem 6.2.1. *The following inclusions hold:*

(a) $\mathbf{SPCF} \subset \mathbf{CF}$

(b) $\mathbf{0L} \subset \mathbf{PPCF}$

(c) *Let* $X \in \{\mathbf{SP}, \mathbf{JSP}, \mathbf{PP}, \mathbf{JPP}, \mathbf{SPCF}, \mathbf{JSPCF}, \mathbf{PPCF}, \mathbf{JPPCF}\}$. *Then,* $X^{-\varepsilon} \subseteq X$.

Proof.

(a) By its definition, **SPCF** characterize the family of languages generated by pure context-free grammars and hence $\mathbf{SPCF} \subset \mathbf{CF}$ by [63, 88].

(b) Clearly, $\mathbf{0L} \subseteq \mathbf{PPCF}$. Let $L = \{a, aab\}$. Observe that for a pure context-free grammar $G = (\{a, b\}, \{a \to aab\}, a)$, it holds that $L = L(G, {}_p\Rightarrow)$ and hence $L \in \mathbf{PPCF}$. Obviously, $L \notin \mathbf{0L}$, because by Definition 3.2.1, every 0L system H such that $L \subseteq L(H)$ must contain a rule with b on its left-hand side and thus $L(H) - L \neq \emptyset$.

(c) Obvious. □

Theorem 6.2.2. SPCF *and* **JSPCF** *are semilinear.*

Proof. Since **CF** is semilinear (see [75]) and $\mathbf{SPCF} \subset \mathbf{CF}$, **SPCF** must be also semilinear. Consider any pure context-free grammar $G = (\Sigma, P, \sigma)$. From the definitions of ${}_s\Rightarrow$ and ${}_j\Rightarrow$ it follows that $\phi(L(G, {}_s\Rightarrow)) = \phi(L(G, {}_j\Rightarrow))$, where $\phi(L)$ is a Parikh map of L. Thus, **JSPCF** is semilinear as well. □

Theorem 6.2.3. $\mathbf{SPCF} \subset \mathbf{PPCF}$

Proof. First, it will be proven that $\mathbf{SPCF} \subseteq \mathbf{PPCF}$. Let $G = (\Sigma, P, \sigma)$ be a pure context-free grammar. From G, construct a pure context-free grammar

$$G' = (\Sigma, P \cup \{a \to a \mid a \in \Sigma\}, \sigma)$$

Now, the following two claims will be proven.

Claim 6.2.4. *Let $w \in \Sigma^*$. If $\sigma \ {}_s{\Rightarrow}^m_{\mathrm{G}} \ w$, then $\sigma \ {}_p{\Rightarrow}^*_{\mathrm{G}'} \ w$.*

Proof. The claim is proved by induction on $m \geq 0$. As σ is the axiom of both G and G′, the induction basis is obvious. Assume that the claim holds for all $0 \leq m \leq k$, where $k \geq 0$.

Let $\sigma \ {}_s{\Rightarrow}^{k+1}_{\mathrm{G}} \ w$. Express $\sigma \ {}_s{\Rightarrow}^{k+1}_{\mathrm{G}} \ w$ as $\sigma \ {}_s{\Rightarrow}^{k}_{\mathrm{G}} \ uav \ {}_s{\Rightarrow}_{\mathrm{G}} \ uxv$, where $u, v, x \in \Sigma^*$, $a \in \Sigma$, $a \to x \in \mathrm{P}$, and $uxv = w$. By the induction hypothesis, there exists a derivation $\sigma \ {}_p{\Rightarrow}^*_{\mathrm{G}'} \ uav$. By the construction of G′ from G, $uav \ {}_p{\Rightarrow}^*_{\mathrm{G}'} \ uxv$. □

Claim 6.2.5. *Let $w \in \Sigma^*$. If $\sigma \ {}_p{\Rightarrow}^m_{\mathrm{G}'} \ w$, then $\sigma \ {}_s{\Rightarrow}^*_{\mathrm{G}} \ w$.*

Proof. The claim is proved by induction on $m \geq 0$. As in Claim 6.2.4, the induction basis is obvious. Assume that the claim holds for all $0 \leq m \leq k$, where $k \geq 0$.

Let $\sigma \ {}_p{\Rightarrow}^{k+1}_{\mathrm{G}'} \ w$. Express $\sigma \ {}_p{\Rightarrow}^{k+1}_{\mathrm{G}'} \ w$ as $\sigma \ {}_p{\Rightarrow}^{k}_{\mathrm{G}'} \ x \ {}_p{\Rightarrow}_{\mathrm{G}'} \ w$, where $x \in \Sigma^*$. Set $n = |x|$. Express x and w as $x = a_1a_2 \dots a_n$ and $w = y_1y_2 \dots y_n$, respectively, where $a_i \in \Sigma$, $y_i \in \Sigma^*$, and $a_i \to y_i \in (\mathrm{P} \cup \{a \to a \mid a \in \Sigma\})$, $1 \leq i \leq n$. Observe that $a_i \neq y_i$ implies $a_i \to y_i \in \mathrm{P}$, for all $1 \leq i \leq n$, and hence $x \ {}_s{\Rightarrow}^*_{\mathrm{G}} \ w$. □

By Claim 6.2.4 and Claim 6.2.5, $\sigma \ {}_s{\Rightarrow}^*_{\mathrm{G}} \ w$ if and only if $\sigma \ {}_p{\Rightarrow}^*_{\mathrm{G}'} \ w$. That is, $\mathrm{L}(\mathrm{G}, {}_s{\Rightarrow}) = \mathrm{L}(\mathrm{G}', {}_p{\Rightarrow})$ and thus **SPCF** $\subseteq$ **PPCF**.

Let $\mathrm{L} = \{a^{2^n} \mid n \geq 0\}$. As L $\notin$ **CF**, L $\notin$ **SPCF**. Let $\mathrm{G} = (\{a\}, \{a \to aa\}, a)$. Clearly, $\mathrm{L} = \mathrm{L}(\mathrm{G}, {}_p{\Rightarrow})$ and thus L $\in$ **PPCF**. Hence, **SPCF** $\subset$ **PPCF**. □

Corollary 6.2.6. SPCF $\subset$ **0L**

Proof. It follows directly from the construction of G′ from G in the proof of Theorem 6.2.3 and from the fact that G′ is 0L system. □

Theorem 6.2.7. SPCF $\subset$ (**CF** $\cap$ **PPCF**)

Proof. For every L $\in$ **SPCF**, it holds that L $\in$ **CF** (Theorem 6.2.1.a) and simultaneously L $\in$ **PPCF** (Theorem 6.2.3), and hence **SPCF** $\subseteq$ (**CF** $\cap$ **PPCF**).

Let $\mathrm{L} = \{ab, ccdd\}$ be a language over $\Sigma = \{a, b, c, d\}$. Clearly, L $\in$ **CF** and also L $\in$ **PPCF** since there is a pure context-free grammar

$$\mathrm{G} = (\Sigma, \{a \to cc, b \to dd, c \to c, d \to d\}, ab)$$

such that $\mathrm{L} = \mathrm{L}(\mathrm{G}, {}_p{\Rightarrow})$.

By contradiction, it will be demonstrated that there is no pure context-free grammar $\mathrm{G}' = (\Sigma, \mathrm{P}', \sigma)$ such that $\mathrm{L}(\mathrm{G}', {}_s{\Rightarrow}) = \mathrm{L}$.

Clearly, σ must be either ab or $ccdd$. If $ccdd$ is chosen as the axiom, there must exist rule $c \to \varepsilon$ or $d \to \varepsilon$ in P′ and hence cdd or ccd are contained in L, which is a contradiction.

On the other hand, if ab is chosen instead, then there is no possible way to directly derive $ccdd$ from ab by using ${}_s\Rightarrow$. Thus, $\mathrm{L} \notin$ **SPCF**, which completes the proof. □

Corollary 6.2.8. SPCF $\subset$ **(CF** $\cap$ **0L)**

Proof. It follows directly from Theorem 6.2.7 and its proof. □

Corollary 6.2.9. *For a unary alphabet,* **0L** = **PPCF** = **JPPCF**.

Proof. It follows directly from the definition of ${}_p\Rightarrow$ and ${}_{jp}\Rightarrow$ and from the definition of $\Rightarrow$ in 0L systems (see Definition 3.2.1). □

Corollary 6.2.10. *For a unary alphabet,* **SPCF** = **JSPCF**.

Proof. It follows directly from the definition of ${}_s\Rightarrow$ and ${}_j\Rightarrow$. □

Corollary 6.2.11. *For a unary alphabet,* **JSPCF** $\subset$ **JPPCF**.

Proof. It follows directly from Corollary 6.2.9, Corollary 6.2.10, and Theorem 6.2.3. □

The following lemma is a generalization of Lemma 3.2.2.

Lemma 6.2.12. *Let* G *be a pure context-free grammar. Let* $h \in \{s, j, p, jp\}$. *Then there exists a number* k *such that for every word* $w \in \mathrm{L}(\mathrm{G}, {}_h\Rightarrow)$ *there exists a derivation such that* $|u| \leq k|w|$ *for every word* u *in that derivation.*

Proof. It follows directly from the proof of Lemma 4.8 in [85]. □

Theorem 6.2.13. CS $-$ **JPPCF** $\neq \emptyset$

Proof. The language $\mathrm{L} = \{a^p \mid p \text{ is a prime}\}$ over a unary alphabet $\{a\}$ is a well-known context-sensitive non-context-free language (see [31]). By using contradiction, it will be shown that $\mathrm{L} \notin$ **JPPCF**. Assume that there is a pure context-free grammar $\mathrm{G} = (\{a\}, \mathrm{P}, \sigma)$ such that $\mathrm{L}(\mathrm{G}, {}_{jp}\Rightarrow) = \mathrm{L}$. Obviously, $a \rightarrow \varepsilon \notin \mathrm{P}$ and $\sigma = a^2$ since 2 is the smallest prime. As 3 is also prime, $a^2 \ {}_{jp}\Rightarrow^*_{\mathrm{G}} a^3$ and then there must be rules $a \rightarrow a \in \mathrm{P}$ and $a \rightarrow a^2 \in \mathrm{P}$. Thus, $a^2 \ {}_{jp}\Rightarrow^*_{\mathrm{G}} a^4$, which is a contradiction because 4 is not a prime. □

Theorem 6.2.14. CS $-$ **JSPCF** $\neq \emptyset$

Proof. Let $\mathrm{L} = \{a^p \mid p \text{ is a prime}\}$. By Theorem 6.2.13, $\mathrm{L} \notin$ **JPPCF**. Since L is a unary language and for unary languages it holds that **JSPCF** $\subset$ **JPPCF** (see Corollary 6.2.11), $\mathrm{L} \notin$ **JSPCF**. □

Theorem 6.2.15. JPPCF $\subset$ **CS**

Proof. Let $\mathrm{G} = (\Sigma, \mathrm{P}, \sigma)$ be a pure context-free grammar. Clearly, there is an unrestricted grammar $\mathrm{H} = (\mathrm{V}, \Sigma, \mathrm{P}', \mathrm{S})$ such that $\mathrm{L}(\mathrm{H}) = \mathrm{L}(\mathrm{G}, {}_{jp}\Rightarrow)$. More precisely, H can be constructed in the way that H simulates G. In this case, Lemma 6.2.12 also holds for H.

Observe that Lemma 6.2.12 is the workspace theorem, and every language from **JPPCF** must be therefore context-sensitive.

By Theorem 6.2.13, **CS** − **JPPCF** ≠ ∅ and hence **JPPCF** ⊂ **CS**. □

Theorem 6.2.16. JSPCF ⊂ CS

Proof. **JSPCF** ⊆ **CS** can be proven analogously as in Theorem 6.2.15. By Theorem 6.2.14, **CS** − **JSPCF** ≠ ∅ and hence **JSPCF** ⊂ **CS**. □

6.3 Relationship with the Chomsky Hierarchy of Languages

This section investigates all mutual relations between **SPCF**, **JSPCF**, **PPCF**, **JPPCF**, **CF**, and **CS**. They are referred to as language subfamilies $\mathcal{A}$ through $\mathcal{T}$ and visualized in Figure 1 using an Euler diagram. More precisely, in this diagram, **JSPCF**, **PPCF**, **JPPCF**, **CF** form a Venn diagram with sixteen subfamilies contained in **CS**; in addition, four more subfamilies are depicted by placing **SPCF** as a subset of **CF** ∩ **PPCF** (see Theorem 6.2.7). Hereafter, twenty subfamilies in the following thirteen theorems and seven open problems (Theorems and Open Problems 6.3.1 through 6.3.20) are studied.

Theorem 6.3.1. *(Subfamily $\mathcal{A}$)*

$$\mathbf{PPCF} - (\mathbf{CF} \cup \mathbf{JSPCF} \cup \mathbf{JPPCF}) \neq \emptyset$$

Proof. Let $\Sigma = \{a, b\}$ be an alphabet. Let $\mathrm{L} = \{a^{2^n} b^{2^n} \mid n \geq 0\}$ be a language over Σ. Clearly, $\mathrm{L} \in$ **PPCF**, since there exists a pure context-free grammar, $\mathrm{G} = (\Sigma, \{a \to aa, b \to bb\}, ab)$, such that $\mathrm{L}(\mathrm{G}, {}_p\!\Rightarrow) = \mathrm{L}$. $\mathrm{L} \notin$ **CF** and $\mathrm{L} \notin$ **JSPCF** is satisfied since L is not semilinear. By contradiction, it will be shown that $\mathrm{L} \notin$ **JPPCF**.

Consider that there is a pure context-free grammar, $\mathrm{G}' = (\Sigma, \mathrm{P}', \sigma')$, such that $\mathrm{L}(\mathrm{G}', {}_{jp}\!\Rightarrow) = \mathrm{L}$. Observe that $ab \in \mathrm{L}(\mathrm{G}', {}_{jp}\!\Rightarrow)$. Let $a \to x$, $b \to y$ be rules from P', $x, y \in \Sigma^*$. Then, there exist two derivations, $ab\ {}_{jp}\!\Rightarrow_{\mathrm{G}'} xy$ and $ab\ {}_{jp}\!\Rightarrow_{\mathrm{G}'} yx$. Now, consider the following cases:

- $x = \varepsilon$ ($y = \varepsilon$). If $y \in \mathrm{L}$ ($x \in \mathrm{L}$), then either ab is the only word derivable using ${}_{jp}\!\Rightarrow_{\mathrm{G}'}$ or there is a derivation $y\ {}_{jp}\!\Rightarrow^*_{\mathrm{G}'} z$ ($x\ {}_{jp}\!\Rightarrow^*_{\mathrm{G}'} z$) such that $ba \in \mathrm{subword}(z)$, which is a contradiction. If $y \notin \mathrm{L}$, such as $y \in \{\varepsilon, a, b\}$ ($x \notin \mathrm{L}$, such as $x \in \{\varepsilon, a, b\}$), then $ab\ {}_{jp}\!\Rightarrow_{\mathrm{G}'} y$ ($ab\ {}_{jp}\!\Rightarrow_{\mathrm{G}'} x$), so $y \in \mathrm{L}$ ($x \in \mathrm{L}$), which is a contradiction as well.

  In the following cases, it is assumed that $x \neq \varepsilon$ and $y \neq \varepsilon$.

- $x = bx'$ or $y = by'$, where $x', y' \in \Sigma^*$. Then, there is a derivation $ab\ {}_{jp}\!\Rightarrow_{\mathrm{G}'} bz$, where $z \in \Sigma^*$, and thus $bz \in \mathrm{L}$, which is a contradiction.
- $x = x'a$ or $y = y'a$, where $x', y' \in \Sigma^*$. Then, there is a derivation $ab\ {}_{jp}\!\Rightarrow_{\mathrm{G}'} za$, where $z \in \Sigma^*$, and thus $za \in \mathrm{L}$, which is a contradiction.

- $x = ax'b$ and $y = ay'b$, where $x', y' \in \Sigma^*$. Then, there is a derivation $ab\ {}_{jp}\!\Rightarrow_{G'} z$ such that $ba \in \text{subword}(z)$, which is a contradiction.

If $a \to x$ or $b \to y$ is missing in P', then L is finite—a contradiction. No other cases are possible, which completes the proof. □

Several intersections of some language families are hard to investigate. Such an intersection is **PPCF** ∩ **JSPCF**. At the moment, it is not known whether **PPCF** ∩ **JSPCF** ⊆ **CF** or not. For this reason, the properties of subfamilies $\mathcal{B}$ and $\mathcal{C}$ are left as open problems.

Open Problem 6.3.2. (Subfamily $\mathcal{B}$) Is it true that

$$(\mathbf{PPCF} \cap \mathbf{JSPCF}) - (\mathbf{CF} \cup \mathbf{JPPCF}) \neq \emptyset?$$

Open Problem 6.3.3. (Subfamily $\mathcal{C}$) Is it true that

$$(\mathbf{PPCF} \cap \mathbf{JSPCF} \cap \mathbf{JPPCF}) - \mathbf{CF} \neq \emptyset?$$

Theorem 6.3.4. *(Subfamily $\mathcal{D}$)*

$$(\mathbf{PPCF} \cap \mathbf{JPPCF}) - (\mathbf{CF} \cup \mathbf{JSPCF}) \neq \emptyset$$

Proof. For unary alphabet, **0L** = **PPCF** = **JPPCF** (Corollary 6.2.9). Since **CF** and **JSPCF** are both semilinear, it is sufficient to find any non-semilinear language over unary alphabet which is also contained in **PPCF**. Such a language is indisputably $\{a^{2^n} \mid n \geq 0\}$. □

Theorem 6.3.5. *(Subfamily $\mathcal{E}$)*

$$\mathbf{SPCF} - (\mathbf{JSPCF} \cup \mathbf{JPPCF}) \neq \emptyset$$

Proof. Let $\Sigma = \{a, b, c\}$ be an alphabet. Let $L = \{a^n cb^n \mid n \geq 0\}$ be a language over Σ. Clearly, there exists a pure context-free grammar $G = (\Sigma, \{c \to acb\}, c)$ such that $L(G, {}_s\!\Rightarrow) = L$ and hence $L \in$ **SPCF**. By contradiction, it will be proven that L is neither jumping sequential pure context-free nor jumping parallel pure context-free language.

$L \notin$ **JSPCF**. Assume that there is a pure context-free grammar $G' = (\Sigma, P', \sigma')$ such that $L(G', {}_j\!\Rightarrow) = L$. Clearly, $\sigma' = c$ must be the axiom since there must be no erasing rules in P' (observe that $ab, ac, cb \notin L$). Because $acb \in L$, there must be a rule $c \to acb \in P'$. However, $acb\ {}_j\!\Rightarrow_{G'} abacb$ and $abacb \notin L$, which is a contradiction.

$L \notin$ **JPPCF**. Assume that there is a pure context-free grammar $H = (\Sigma, R, \omega)$ such that $L(H, {}_{jp}\!\Rightarrow) = L$. First, let $k \geq 1$ and assume that $\omega = a^k cb^k$ is an axiom. Since $\omega\ {}_{jp}\!\Rightarrow^*_H c$, there must be a rules $a \to \varepsilon$, $b \to \varepsilon$, and $c \to c$ contained in R. Now, assume that

- $\hat{d} \to dx \in R$, $\hat{d} \in \{a, b\}$, $d \in \Sigma$, $x \in \Sigma^*$; then, $\omega\ {}_{jp}\!\Rightarrow^*_H udxcv$ and $\omega\ {}_{jp}\!\Rightarrow^*_H ucdxv$ and obviously $d = a$ implies $ucdxv \notin L$ and $d = b$ implies $udxcv \notin L$, $u, v \in \Sigma^*$; $d = c$ is obvious;

- $\hat{d} \rightarrow xd \in \mathrm{R}$, $\hat{d} \in \{a, b\}$, $d \in \Sigma$, $x \in \Sigma^*$; then, $\omega \ {}_{jp}{\Rightarrow}_{\mathrm{H}}^{*} \ uxdcv$ and $\omega \ {}_{jp}{\Rightarrow}_{\mathrm{H}}^{*} \ ucxdv$ and obviously $d = a$ implies $ucxdv \notin \mathrm{L}$ and $d = b$ implies $uxdcv \notin \mathrm{L}$, $u, v \in \Sigma^*$; $d = c$ is obvious.

Therefore, $a \rightarrow x, b \rightarrow y \in \mathrm{R}$ implies $x = y = \varepsilon$. Hence, only rules of the form $c \rightarrow z$, where $z \in \mathrm{L}$, can be considered. However, the finiteness of R implies the finiteness of L, which is a contradiction.

Clearly, the axiom must be $\omega = c$, which implies that R contains rules of the form $c \rightarrow z$, where $z \in \mathrm{L}$. Obviously, there must be also rules $a \rightarrow x, b \rightarrow y \in \mathrm{R}$, $x, y \in \Sigma^*$. If $x = y = \varepsilon$, L must be finite. Thus, assume that $x \neq \varepsilon$ or $y \neq \varepsilon$. Then, like before, a word which is not contained in L can be derived—a contradiction. □

Theorem 6.3.6. *(Subfamily $\mathcal{F}$)*

$$(\mathbf{SPCF} \cap \mathbf{JSPCF}) - \mathbf{JPPCF} \neq \emptyset$$

Proof. Let $\Sigma = \{a, b, c\}$ be an alphabet and let $\mathrm{L} = \{aa, aab, aac, aabc\}$ be a language over Σ. Consider a pure context-free grammar

$$\mathrm{G} = (\Sigma, \{b \rightarrow \varepsilon, c \rightarrow \varepsilon\}, aabc)$$

Clearly, $\mathrm{L}(\mathrm{G}, {}_{s}{\Rightarrow}) = \mathrm{L}(\mathrm{G}, {}_{j}{\Rightarrow}) = \mathrm{L}$ and hence $\mathrm{L} \in (\mathbf{SPCF} \cap \mathbf{JSPCF})$.

By contradiction, it will be shown that $\mathrm{L} \notin \mathbf{JPPCF}$. Assume that there exists a pure context-free grammar $\mathrm{G}' = (\Sigma, \mathrm{P}', \sigma)$ such that $\mathrm{L}(\mathrm{G}', {}_{jp}{\Rightarrow}) = \mathrm{L}$. Since $\sigma \in \mathrm{L}$ and $\mathrm{L} \subseteq \{aa\}\{b\}^*\{c\}^*$, there must be a rule $a \rightarrow x \in \mathrm{P}'$ with $x \in \Sigma^*$. However, this implies that there must be a derivation $\sigma \ {}_{jp}{\Rightarrow}_{\mathrm{G}'}^{*} \ aa \ {}_{jp}{\Rightarrow}_{\mathrm{G}'} \ xx$. The only word from L that has a form xx is aa so $a \rightarrow a$ is the only rule with a on its left-hand side so $a \rightarrow a \in \mathrm{P}'$.

As the next step, choose σ. Clearly, $\sigma \neq aa$. Furthermore, $\sigma \notin \{aab, aac\}$ since $\sigma \ {}_{jp}{\Rightarrow}_{\mathrm{G}'} \ aabc$ implies that $\sigma \ {}_{jp}{\Rightarrow}_{\mathrm{G}'}^{*} \ abca$, and $abca \notin \mathrm{L}$. Thus, the only possibility is to choose $\sigma = aabc$. However, $aabc \ {}_{jp}{\Rightarrow}_{\mathrm{G}'} \ aab$ means that $\{b \rightarrow b, c \rightarrow \varepsilon\} \subseteq \mathrm{P}'$ or $\{b \rightarrow \varepsilon, c \rightarrow b\} \subseteq \mathrm{P}'$. In both cases, $aabc \ {}_{jp}{\Rightarrow}_{\mathrm{G}'} \ aba$. As $aba \notin \mathrm{L}$, there is no pure context-free grammar G' such that $\mathrm{L}(\mathrm{G}', {}_{jp}{\Rightarrow}) = \mathrm{L}$, which is a contradiction. □

Theorem 6.3.7. *(Subfamily $\mathcal{G}$)*

$$\mathbf{SPCF} \cap \mathbf{JSPCF} \cap \mathbf{JPPCF} \neq \emptyset$$

Proof. Let $\mathrm{G} = (\{a\}, \{a \rightarrow a, a \rightarrow aa\}, a)$ be a pure context-free grammar. It is easy to see that

$$\mathrm{L}(\mathrm{G}, {}_{s}{\Rightarrow}) = \mathrm{L}(\mathrm{G}, {}_{j}{\Rightarrow}) = \mathrm{L}(\mathrm{G}, {}_{jp}{\Rightarrow}) = \{a\}^+$$

□

Open Problem 6.3.8. (Subfamily $\mathcal{H}$) Is it true that

$$(\mathbf{SPCF} \cap \mathbf{JPPCF}) - \mathbf{JSPCF} \neq \emptyset?$$

Theorem 6.3.9. *(Subfamily $\mathcal{I}$)*

$$(\mathbf{PPCF} \cap \mathbf{CF}) - (\mathbf{SPCF} \cup \mathbf{JSPCF} \cup \mathbf{JPPCF}) \neq \emptyset$$

Proof. Let $L = \{aabb, ccdd\}$ be a language over an alphabet $\Sigma = \{a, b, c, d\}$. Clearly, $L \in \mathbf{CF}$. Since there exists a pure context-free grammar $G = (\Sigma, \{a \to c, b \to d\}, aabb)$ such that $L(G, {}_p\Rightarrow) = L$, $L \in \mathbf{PPCF}$. Furthermore, observe that derivations $aabb\ {}_s\Rightarrow\ ccdd$ ($aabb\ {}_j\Rightarrow\ ccdd$) or $ccdd\ {}_s\Rightarrow\ aabb$ ($ccdd\ {}_j\Rightarrow\ aabb$) cannot be performed due to the definition of ${}_s\Rightarrow$ (${}_j\Rightarrow$) and hence there is no pure context-free grammar G' such that $L(G', {}_s\Rightarrow) = L$ ($L(G', {}_j\Rightarrow) = L$). Thus, $L \notin \mathbf{SPCF}$ and $L \notin \mathbf{JSPCF}$.

Now, suppose that there is a pure context-free grammar $H = (\Sigma, P, \sigma)$ such that

$$L(H, {}_{jp}\Rightarrow) = L$$

For $\sigma = aabb$, there is a derivation $aabb\ {}_{jp}\Rightarrow_H\ ccdd$. If $a \to \varepsilon \in P$ or $b \to \varepsilon \in P$, then $aabb\ {}_{jp}\Rightarrow_H\ x$, where $x \notin L$. Thus, $a \to y$ and $b \to z$, where $y, z \in \{c, d\}$, are only possible rules in P. However, $aabb\ {}_{jp}\Rightarrow_H\ cdcd$, and since $cdcd \notin L$, there is no pure context-free grammar H such that $L(H, {}_{jp}\Rightarrow) = L$. The same concept applies to $\sigma = ccdd$. Therefore, $L \notin \mathbf{JPPCF}$. □

Open Problem 6.3.10. (Subfamily $\mathcal{J}$) Is it true that

$$(\mathbf{PPCF} \cap \mathbf{CF} \cap \mathbf{JSPCF}) - (\mathbf{SPCF} \cup \mathbf{JPPCF}) \neq \emptyset?$$

Open Problem 6.3.11. (Subfamily $\mathcal{K}$) Is it true that

$$(\mathbf{PPCF} \cap \mathbf{CF} \cap \mathbf{JSPCF} \cap \mathbf{JPPCF}) - \mathbf{SPCF} \neq \emptyset?$$

Theorem 6.3.12. *(Subfamily $\mathcal{L}$)*

$$(\mathbf{PPCF} \cap \mathbf{CF} \cap \mathbf{JPPCF}) - (\mathbf{SPCF} \cup \mathbf{JSPCF}) \neq \emptyset$$

Proof. Consider a language $L = \{ab, cd, dc\}$ over an alphabet $\Sigma = \{a, b, c, d\}$. Clearly, L is neither a classical sequential pure context-free nor jumping sequential pure context-free language since at some point during a derivation, two symbols must be rewritten simultaneously.

As L is a finite language, $L \in \mathbf{CF}$. As there exists a pure context-free grammar

$$G = (\Sigma, \{a \to c, b \to d, c \to d, d \to c\}, ab)$$

such that $L(G, {}_p\Rightarrow) = L(G, {}_{jp}\Rightarrow) = L$, $L \in (\mathbf{PPCF} \cap \mathbf{JPPCF})$. □

Theorem 6.3.13. *(Subfamily $\mathcal{M}$)*

$$\mathbf{CF} - (\mathbf{PPCF} \cup \mathbf{JSPCF} \cup \mathbf{JPPCF}) \neq \emptyset$$

Proof. Let $\Sigma = \{a, b\}$ and let $L = \{a^n b^n \mid n \geq 1\}$ be a language over Σ. Indisputably, L is a well-known context-free language. According to [85], $L \notin \mathbf{0L}$. Observe that every language L' that belongs to $(\mathbf{PPCF} - \mathbf{0L})$ can be generated by pure context-free grammar $\mathrm{G} = (\Sigma, \mathrm{P}, \sigma)$ where exists $c \in \Sigma$ such that for every $x \in \Sigma^*$, $c \rightarrow x \notin \mathrm{P}$. Thus, if $\mathrm{L} \in (\mathbf{PPCF} - \mathbf{0L})$, then L must be a finite language (since either a or b blocks deriving of any word from axiom), which is a contradiction. Therefore, $\mathrm{L} \notin (\mathbf{PPCF} - \mathbf{0L})$ and clearly $\mathrm{L} \notin \mathbf{PPCF}$. Next, it will be demonstrated that $\mathrm{L} \notin \mathbf{JSPCF}$ and $\mathrm{L} \notin \mathbf{JPPCF}$.

$\mathrm{L} \notin \mathbf{JSPCF}$. Suppose that $\mathrm{L} \in \mathbf{JSPCF}$, so there exists a pure context-free grammar $\mathrm{G}' = (\Sigma, \mathrm{P}', \sigma')$ such that $\mathrm{L}(\mathrm{G}', {}_j\Rightarrow) = \mathrm{L}$. As $a, b \notin \mathrm{L}$, there are no erasing rules in P' and thus $\sigma' = ab$ must be the axiom. Now consider a derivation $ab \ {}_j\Rightarrow_{\mathrm{G}'} aabb$. There are exactly two possibilities how to get the word $aabb$ directly from the axiom ab—either expand a to aab ($a \rightarrow aab \in \mathrm{P}'$) or expand b to abb ($b \rightarrow abb \in \mathrm{P}'$). Due to the definition of ${}_j\Rightarrow$, $ab \ {}_j\Rightarrow_{\mathrm{G}'} baab$ in the first case, and $ab \ {}_j\Rightarrow_{\mathrm{G}'} abba$ in the second case. Since neither $baab$ nor $abba$ belong to L, $\mathrm{L} \notin \mathbf{JSPCF}$, which is a contradiction.

$\mathrm{L} \notin \mathbf{JPPCF}$. Suppose that $\mathrm{L} \in \mathbf{JPPCF}$, so there exists a pure context-free grammar $\mathrm{H} = (\Sigma, \mathrm{R}, \omega)$ such that $\mathrm{L}(\mathrm{H}, {}_{jp}\Rightarrow) = \mathrm{L}$. As for all $k \geq 0$, $a^k, b^k \notin \mathrm{L}$, there are no erasing rules in R and thus $\omega = ab$ must be the axiom. Clearly, $ab \ {}_{jp}\Rightarrow_{\mathrm{H}} aabb$. There are exactly three ways how to get $aabb$ from ab:

- $a \rightarrow a \in \mathrm{R}$, $b \rightarrow abb \in \mathrm{R}$. In this case $ab \ {}_{jp}\Rightarrow_{\mathrm{H}} aabb$ implies that $ab \ {}_{jp}\Rightarrow_{\mathrm{H}} abba$, but $abba \notin \mathrm{L}$.
- $a \rightarrow aa \in \mathrm{R}$, $b \rightarrow bb \in \mathrm{R}$. In this case $ab \ {}_{jp}\Rightarrow_{\mathrm{H}} aabb$ implies that $ab \ {}_{jp}\Rightarrow_{\mathrm{H}} bbaa$, but $bbaa \notin \mathrm{L}$.
- $a \rightarrow aab \in \mathrm{R}$, $b \rightarrow b \in \mathrm{R}$. In this case $ab \ {}_{jp}\Rightarrow_{\mathrm{H}} aabb$ implies that $ab \ {}_{jp}\Rightarrow_{\mathrm{H}} baab$, but $baab \notin \mathrm{L}$.

Thus, $\mathrm{L} \notin \mathbf{JPPCF}$, which is a contradiction. □

Open Problem 6.3.14. (Subfamily $\mathcal{N}$) Is it true that

$$(\mathbf{CF} \cap \mathbf{JSPCF}) - (\mathbf{PPCF} \cup \mathbf{JPPCF}) \neq \emptyset?$$

Theorem 6.3.15. *(Subfamily $\mathcal{O}$)*

$$(\mathbf{CF} \cap \mathbf{JSPCF} \cap \mathbf{JPPCF}) - \mathbf{PPCF} \neq \emptyset$$

Proof. Let $\Sigma = \{a, b\}$ be an alphabet and let

$$\mathrm{L} = \{aabb, abab, abba, baab, baba, bbaa\}$$

be a language over Σ. Since L is finite, L is context-free. Given a pure context-free grammar

$$\mathrm{G} = (\Sigma, \{a \rightarrow a, b \rightarrow b\}, aabb)$$

Clearly, $\mathrm{L}(\mathrm{G}, {}_j\Rightarrow) = \mathrm{L}(\mathrm{G}, {}_{jp}\Rightarrow) = \mathrm{L}$. Hence, $\mathrm{L} \in (\mathbf{CF} \cap \mathbf{JSPCF} \cap \mathbf{JPPCF})$.

By contradiction, it will be shown that $\mathrm{L} \notin \mathbf{PPCF}$. Assume that there is a pure context-free grammar $\mathrm{H} = (\Sigma, \mathrm{P}, \sigma)$ such that $\mathrm{L}(\mathrm{H}, {}_{p}\Rightarrow) = \mathrm{L}$. First, it will be shown that there are no erasing rules in P:

- If $a \to \varepsilon \in \mathrm{P}$ and $b \to \varepsilon \in \mathrm{P}$, then $\varepsilon \in \mathrm{L}$, which is a contradiction.
- If $a \to \varepsilon \in \mathrm{P}$, then $b \to x \in \mathrm{P}$ implies that $x \in \{aa, bb, ab, ba\}$ because for every $w \in \mathrm{L}$, $|w| = 4$. Clearly, if $b \to aa \in \mathrm{P}$, then $aaaa \in \mathrm{L}$, and if $b \to bb \in \mathrm{P}$, then $bbbb \in \mathrm{L}$. As obvious, both cases present a contradiction. On the other hand, if there are no rules in P starting from b apart from $b \to ab$ and/or $b \to ba$, then $aabb \notin \mathrm{L}$, which is a contradiction. Similarly for $b \to \varepsilon \in \mathrm{P}$.

Since all words in L have the same length and there are no erasing rules in P, only unit rules can be contained in P. Because $aaaa \notin \mathrm{L}$ and $bbbb \notin \mathrm{L}$, either $\mathrm{P} = \{a \to a, b \to b\}$ or $\mathrm{P} = \{a \to b, b \to a\}$. In both cases, there is no way to achieve L. Thus, there is no pure context-free grammar H such that $\mathrm{L}(\mathrm{H}, {}_{p}\Rightarrow) = \mathrm{L}$, and hence $\mathrm{L} \notin \mathbf{PPCF}$. □

Theorem 6.3.16. *(Subfamily $\mathcal{P}$)*

$$(\mathbf{CF} \cap \mathbf{JPPCF}) - (\mathbf{PPCF} \cup \mathbf{JSPCF}) \neq \emptyset$$

Proof. Consider a language

$$\mathrm{L}' = \{aabb, ccdd, cdcd, cddc, dccd, dcdc, ddcc\}$$

over an alphabet $\Sigma = \{a, b, c, d\}$. Clearly, $\mathrm{L}' \in \mathbf{CF}$ and also $\mathrm{L}' \in \mathbf{JPPCF}$ because there is a pure context-free grammar

$$\mathrm{G} = (\Sigma, \{a \to c, b \to d, c \to c, d \to d\}, aabb)$$

such that $\mathrm{L}(\mathrm{G}, {}_{jp}\Rightarrow) = \mathrm{L}'$. To prove that $\mathrm{L}' \notin \mathbf{PPCF}$, follow the steps of the proof that $\mathrm{L} \notin \mathbf{PPCF}$ of Theorem 6.3.15. Because it is not possible to rewrite two or more symbols simultaneously during a direct derivation step by using ${}_{j}\Rightarrow$, it holds that $\mathrm{L}' \notin \mathbf{JSPCF}$. □

Open Problem 6.3.17. (Subfamily $\mathcal{Q}$) Is it true that

$$\mathbf{JSPCF} - (\mathbf{CF} \cup \mathbf{PPCF} \cup \mathbf{JPPCF}) \neq \emptyset?$$

Theorem 6.3.18. *(Subfamily $\mathcal{R}$)*

$$(\mathbf{JSPCF} \cap \mathbf{JPPCF}) - (\mathbf{CF} \cup \mathbf{PPCF}) \neq \emptyset$$

Proof. Let $\Sigma = \{a, b, c\}$ be an alphabet and let

$$\mathrm{L} = \{w \mid \mathscr{O}_a(w) - 1 = \mathscr{O}_b(w) = \mathscr{O}_c(w), w \in \Sigma^+\}$$

be a language over Σ. $\mathrm{L} \in (\mathbf{JSPCF} \cap \mathbf{JPPCF})$ since there is a pure contex-free grammar

$$\mathrm{G} = (\Sigma, \{a \to abca, a \to a, b \to b, c \to c\}, a)$$

such that $L(G, {}_j\Rightarrow) = L(G, {}_{jp}\Rightarrow) = L$. By pumping lemma for context-free languages, $L \notin \mathbf{CF}$.

By contradiction, it will be shown that $L \notin \mathbf{PPCF}$. Assume that there is a pure context-free grammar $H = (\Sigma, P, \sigma)$ such that $L(H, {}_p\Rightarrow) = L$. As a first step, it will be shown that $\sigma = a$. Assume that $\sigma \neq a$. Then, $\sigma \; {}_p\Rightarrow^*_H \; a$ implies that $a \rightarrow \varepsilon \in P$ and hence $\varepsilon \in L$, which is a contradiction. Thus, a must be the axiom, and $a \rightarrow x \in P$ implies that $x \in L$.

Let $l = 3 \max\{|\beta| \mid \alpha \rightarrow \beta \in P\}$. The smallest possible value of l is 3. Let $\omega = a^{l+1}b^l c^l$. Clearly, $\omega \in L$. Then there is a direct derivation step $\theta \; {}_p\Rightarrow_H \; \omega$, where $\theta \in L$. Next, the following observations about θ and P will be made:

(1) $\theta \neq a$, since $a \rightarrow \omega \notin P$. The choice of l excludes such situation.

(2) θ contains all three symbols a, b, and c.

(3) $a \rightarrow a \in P$ is the only rule with a on its left-hand side that is used during $\theta \; {}_p\Rightarrow_H \; \omega$. Observe that if $a \rightarrow x \in P$ is chosen to rewrite a during $\theta \; {}_p\Rightarrow_H \; \omega$, then $x \in L$ and x must be a subword of ω. Only $x = a$ meets these requirements.

(4) θ can be expressed as $a^+\theta'$, where $\theta' \in \{b, c\}^*$. This follows from the form of ω and the observation (3).

(5) During $\theta \; {}_p\Rightarrow_H \; \omega$ rules $b \rightarrow y, c \rightarrow y' \in P$ are used such that each of y, y' do not contain at least one symbol from Σ. This is secured by the choice of l.

(6) Every rule with b on its left-hand side in P has the same commutative image of its right-hand side and every rule with c on its left-hand side in P has the same commutative image of its right-hand side. To not break the number of occurrences of symbols a, b, and c in ω during $\theta \; {}_p\Rightarrow_H \; \omega$, when $b \rightarrow y \in P$ is used, the corresponding $c \rightarrow y' \in P$ must be also used simultaneously. To preserve the proper number of occurrences of a, b, and c in ω, it holds that

$$\mathrm{card}(\{\psi(\beta) \mid b \rightarrow \beta \in P\}) = 1 \quad \text{and} \quad \mathrm{card}(\{\psi(\gamma) \mid c \rightarrow \gamma \in P\}) = 1$$

where $\psi(w)$ denotes a commutative image of w.

Now, the ways how $a^+\theta' \; {}_p\Rightarrow_H \; \omega$ could be made will be inspected. Suppose that the first symbol of θ' is b:

- $b \rightarrow \varepsilon \in P$ was used. Then, $c \rightarrow bc \in P$ must be used ($c \rightarrow cb$ is excluded since c is not before b in ω). As there are at least two cs in θ', applying $c \rightarrow bc$ brings c before b which is in a contradiction with the form of ω.
- Let $i \geq 1$ and let $b \rightarrow a^i \in P$. Then, $c \rightarrow b^{i+1}c^{i+1} \in P$. Since $|b^{i+1}c^{i+1}|$ is at most $\frac{l}{3}$, there are at least two occurrences of c in θ' and c appears before b in ω.
- Let $i \geq 1$ and let j be a non-negative integer such that $j \leq i + 1$. Let $b \rightarrow a^i b^j \in P$. Then $c \rightarrow b^k c^m \in P$, where $j + k = m = i + 1$. As in the previous case, when these rules are used during $\theta \; {}_p\Rightarrow_H \; \omega$, b appears before a or c appears before b in ω.

- No as were added during $\theta \ {}_p\!\Rightarrow_{\mathrm{H}} \omega$. In this case, the only rules with b and c on their left-hand sides in P can be either $b \to bc$ and $c \to \varepsilon$, or $b \to b$ and $c \to c$, or $b \to c$ and $c \to b$. This implies that the only way how to get θ from a is to use the $a \to \theta$ rule that is clearly not in P.

The case that c is the first symbol of θ' proceeds analogously. Therefore, $\omega \notin \mathrm{L}(\mathrm{H}, {}_p\!\Rightarrow)$, which implies that $\mathrm{L} \notin \mathbf{PPCF}$. □

Theorem 6.3.19. *(Subfamily $\mathcal{S}$)*

$$\mathbf{JPPCF} - (\mathbf{CF} \cup \mathbf{PPCF} \cup \mathbf{JSPCF}) \neq \emptyset$$

Proof. Let $\Sigma = \{a, b, c, \hat{a}, \hat{b}, \hat{c}\}$ be an alphabet and let

$$\mathrm{L} = \{\hat{a}\hat{b}\hat{c}\} \cup \{w \mid \mathscr{O}_a(w) - 1 = \mathscr{O}_b(w) = \mathscr{O}_c(w), w \in \{a, b, c\}^+\}$$

be a language over Σ. Following the pumping lemma for context-free languages, $\mathrm{L} \notin \mathbf{CF}$. Since there is a pure context-free grammar $\mathrm{G} = (\Sigma, \mathrm{R}, \hat{a}\hat{b}\hat{c})$, where

$$\mathrm{R} = \{\hat{a} \to a, \hat{b} \to \varepsilon, \hat{c} \to \varepsilon, a \to abca, a \to a, b \to b, c \to c\}$$

such that $\mathrm{L}(\mathrm{G}, {}_{jp}\!\Rightarrow) = \mathrm{L}$, $\mathrm{L} \in \mathbf{JPPCF}$. By contradiction, it will be shown that $\mathrm{L} \notin \mathbf{JSPCF}$ and $\mathrm{L} \notin \mathbf{PPCF}$.

Suppose that $\mathrm{L} \in \mathbf{JSPCF}$. Then, there is a pure context-free grammar $\mathrm{H} = (\Sigma, \mathrm{P}, \sigma)$ such that $\mathrm{L}(\mathrm{H}, {}_j\!\Rightarrow) = \mathrm{L}$. As the first step, choose σ. From the definition of L, $a \in \mathrm{L}$ and every word $x \in \mathrm{L} - \{a\}$ holds $|x| \geq 3$. Since only one symbol can be erased during direct derivation step by ${}_j\!\Rightarrow$ and there is no word of length 2 contained in L, $\sigma = a$ must be chosen as the axiom. Because $abca \in \mathrm{L}$ and $\hat{a}\hat{b}\hat{c} \in \mathrm{L}$, there must be two derivations, $a \ {}_j\!\Rightarrow^*_{\mathrm{H}} abca$ and $a \ {}_j\!\Rightarrow^*_{\mathrm{H}} \hat{a}\hat{b}\hat{c}$, which implies that there exists also a derivation $a \ {}_j\!\Rightarrow^*_{\mathrm{H}} \hat{a}\hat{b}\hat{c}bca$. However, $\hat{a}\hat{b}\hat{c}bca \notin \mathrm{L}$, which is a contradiction.

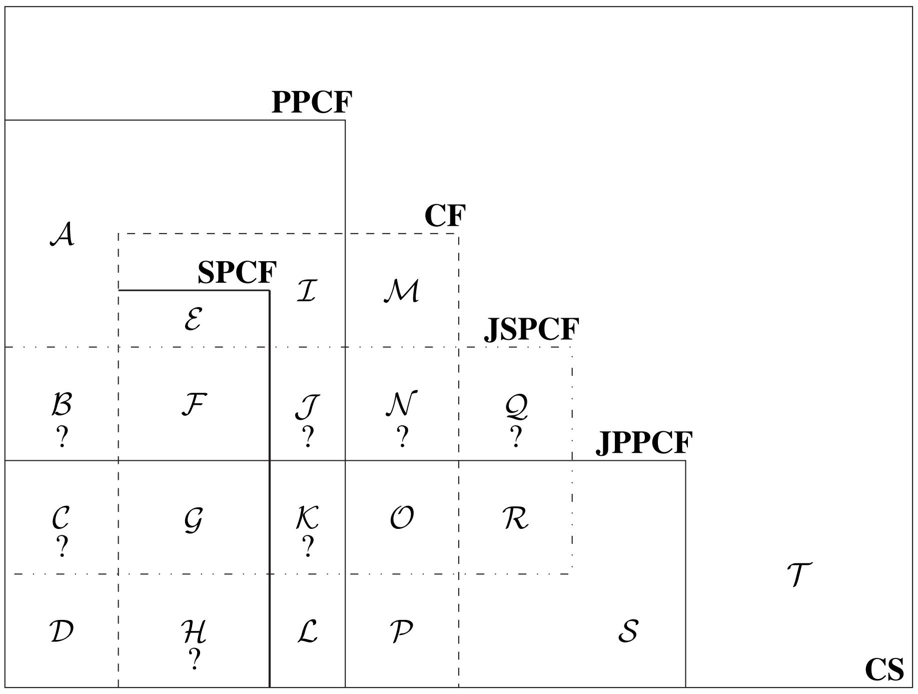
Next, suppose that $\mathrm{L} \in \mathbf{PPCF}$, so there exists a pure context-free grammar $\mathrm{H}' = (\Sigma, \mathrm{P}', \sigma')$ such that $\mathrm{L}(\mathrm{H}', {}_p\!\Rightarrow) = \mathrm{L}$. In this case, $\sigma' = \hat{a}\hat{b}\hat{c}$ must be chosen as the axiom. If a is chosen, then $a \ {}_p\!\Rightarrow^*_{\mathrm{H}'} abca$ and $a \ {}_p\!\Rightarrow^*_{\mathrm{H}'} \hat{a}\hat{b}\hat{c}$ implies that $a \ {}_p\!\Rightarrow^*_{\mathrm{H}'} u_1 a u_2 \hat{a} u_3$, $u_1, u_2, u_3 \in \Sigma^*$, and $u_1 a u_2 \hat{a} u_3 \notin \mathrm{L}$. If $abca$ or similar is chosen, then $abca \ {}_p\!\Rightarrow^*_{\mathrm{H}'} a$ implies that $a \ {}_p\!\Rightarrow^*_{\mathrm{H}'} \varepsilon$, and $\varepsilon \notin \mathrm{L}$. Without loss of generality, assume that for every $\alpha \to \beta \in \mathrm{P}'$, $\beta \in \{a, b, c\}^*$ (this can be assumed since $\hat{a}\hat{b}\hat{c}$ is the only word over $\{\hat{a}, \hat{b}, \hat{c}\}$ in L). As $a \in \mathrm{L}$, $a \to \varepsilon$, $a \to b$, and $a \to c$ are not contained in P'. The observations (1) to (3) from the proof of Theorem 6.3.18 hold also for H'. The rest of proof is similar to the proof of Theorem 6.3.18. □

Theorem 6.3.20. *(Subfamily $\mathcal{T}$)*

$$\mathbf{CS} - (\mathbf{CF} \cup \mathbf{JSPCF} \cup \mathbf{PPCF} \cup \mathbf{JPPCF}) \neq \emptyset$$

Proof. Let $\mathrm{L} = \{a^p \mid p \text{ is a prime}\}$ be a language over unary alphabet $\{a\}$. $\mathrm{L} \in \mathbf{CS}$ and $\mathrm{L} \notin \mathbf{CF}$ are well-known containments (see [31]). By Theorem 6.2.13 and Theorem 6.2.14, $\mathrm{L} \notin \mathbf{JPPCF}$ and $\mathrm{L} \notin \mathbf{JSPCF}$. As for unary languages $\mathbf{PPCF} = \mathbf{JPPCF}$, $\mathrm{L} \notin \mathbf{PPCF}$. □

The summary of theorems 6.3.1 through 6.3.20 is visualized in Figure 1.



$\{a^{2^n}b^{2^n} \mid n \geq 0\} \in \mathcal{A}$	$\{a^n b^n \mid n \geq 1\} \in \mathcal{M}$
$\{a^{2^n} \mid n \geq 0\} \in \mathcal{D}$	$\{aabb, abab, abba, baab, baba, bbaa\} \in \mathcal{O}$
$\{a^n c b^n \mid n \geq 0\} \in \mathcal{E}$	$\{aabb, ccdd, cdcd, cddc, dccd, dcdc, ddcc\} \in \mathcal{P}$
$\{aa, aab, aac, aabc\} \in \mathcal{F}$	$\left\{ w \,\middle|\, \begin{array}{l} \mathscr{O}_a(w) - 1 = \mathscr{O}_b(w) = \mathscr{O}_c(w) \\ w \in \{a, b, c\}^+ \end{array} \right\} \in \mathcal{R}$
$\{a\}^+ \in \mathcal{G}$	$\{\hat{a}\hat{b}\hat{c}\} \cup \left\{ w \,\middle|\, \begin{array}{l} \mathscr{O}_a(w) - 1 = \mathscr{O}_b(w) = \mathscr{O}_c(w) \\ w \in \{a, b, c\}^+ \end{array} \right\} \in \mathcal{S}$
$\{aabb, ccdd\} \in \mathcal{I}$	$\{a^p \mid p \text{ is a prime}\} \in \mathcal{T}$
$\{ab, cd, dc\} \in \mathcal{L}$	

Figure 1. Summary of hierarchy between **SPCF**, **JSPCF**, **PPCF**, **JPPCF**, **CF**, and **CS** language families (? stands for an open problem of the existence of a witness language).

6.4 Remarks on Propagating and Unary Case

As stated in Theorem 6.2.1.c, it is natural that the family of languages generated by propagating pure grammars is included in the family of languages generated by pure grammars in which the presence of erasing rules is allowed. For

$$\mathrm{X} \in \{\mathbf{SP}, \mathbf{PP}, \mathbf{JSP}, \mathbf{JPP}, \mathbf{SPCF}, \mathbf{PPCF}, \mathbf{JSPCF}, \mathbf{JPPCF}\}$$

it is easy to see that a language $\{\varepsilon, a\}$ is contained in X but it is not contained in $\mathrm{X}^{-\varepsilon}$. Therefore, given a language L, the more interesting question is whether $\mathrm{L} - \{\varepsilon\}$ can be generated by both a pure grammar and a propagating pure grammar under a specific derivation mode. As will be shown further, such a question can be answered positively for pure context-free grammars but it is left as an open problem for pure grammars in general.

Theorem 6.4.1. *Let* $\mathrm{X} \in \{\mathbf{SPCF}, \mathbf{JSPCF}, \mathbf{PPCF}, \mathbf{JPPCF}\}$. *Then,* $\mathrm{X}^{-\varepsilon} \subset \mathrm{X}$.

Proof. Let $L_1 = \{a, ab\}$ and $L_2 = \{aa, aab\}$ be two languages over $\Sigma = \{a, b\}$. Furthermore, let $G = (\Sigma, \{a \to a, b \to \varepsilon\}, ab)$ and $G' = (\Sigma, \{a \to a, b \to \varepsilon\}, aab)$ be two pure context-free grammars.

(a) $\mathbf{SPCF}^{-\varepsilon} \subset \mathbf{SPCF}$. Since $L_1 = L(G, {}_s\Rightarrow)$, $L_1 \in \mathbf{SPCF}$. Assume that $L_1 \in \mathbf{SPCF}^{-\varepsilon}$; then, there is a propagating pure context-free grammar

$$H = (\Sigma, P, \sigma)$$

such that $L(H, {}_s\Rightarrow) = L_1$. Obviously, $\sigma = a$, so $a \to ab \in P$. Hence, $a \ {}_s\Rightarrow_H^* abb$ and since $abb \notin L_1$, $L_1 \notin \mathbf{SPCF}^{-\varepsilon}$.

(b) $\mathbf{JSPCF}^{-\varepsilon} \subset \mathbf{JSPCF}$. $L_1 \in \mathbf{JSPCF}$ and $L_1 \notin \mathbf{JSPCF}^{-\varepsilon}$ are proved analogously to (a).

(c) $\mathbf{PPCF}^{-\varepsilon} \subset \mathbf{PPCF}$. Since $L_2 = L(G', {}_p\Rightarrow)$, $L_2 \in \mathbf{PPCF}$. Assume that $L_2 \in \mathbf{PPCF}^{-\varepsilon}$, so there is a propagating pure context-free grammar

$$H = (\Sigma, P, \sigma)$$

such that $L(H, {}_p\Rightarrow) = L_2$. Obviously, $\sigma = aa$ and then either $\{a \to a, a \to ab\} \subseteq P$ or $\{a \to aa, a \to b\} \subseteq P$. Hence, $aa \ {}_p\Rightarrow_H^* abab$ or $aa \ {}_p\Rightarrow_H^* a^4$ and since $\{abab, a^4\} \cap L_2 = \emptyset$, $L_2 \notin \mathbf{PPCF}^{-\varepsilon}$.

(d) $\mathbf{JPPCF}^{-\varepsilon} \subset \mathbf{JPPCF}$. $L_2 \in \mathbf{JPPCF}$ and $L_2 \notin \mathbf{JPPCF}^{-\varepsilon}$ are proved analogously to (c). □

Open Problem 6.4.2. Let $X \in \{\mathbf{SP}, \mathbf{JSP}, \mathbf{PP}, \mathbf{JPP}\}$. Is the inclusion $X^{-\varepsilon} \subseteq X$, in fact, proper?

Most of the relations between investigated language families (even for those which are generated by propagating pure context-free grammars—the most of languages used in Figure 1 have this property) can be derived from Figure 1 and from mentioned theorems. The following theorems cover the rest.

Theorem 6.4.3. *$\mathbf{SPCF}$ and $\mathbf{PPCF}^{-\varepsilon}$ are incomparable, but not disjoint.*

Proof. Let $L = \{aa, aab\}$ be a language over alphabet $\Sigma = \{a, b\}$. Obviously, there is a pure context-free grammar $G = (\Sigma, \{a \to a, b \to \varepsilon\}, aab)$ such that $L(G, {}_s\Rightarrow) = L$, so $L \in \mathbf{SPCF}$. By Theorem 6.4.1, $L \notin \mathbf{PPCF}^{-\varepsilon}$. Conversely, there is a language $L' = \{a^{2^n} \mid n \geq 0\}$ over $\{a\}$ such that $L' \notin \mathbf{SPCF}$ and $L' \in \mathbf{PPCF}^{-\varepsilon}$ (see $\mathcal{D}$ in Figure 1 and observe that to get L' no erasing rules are needed). Finally, $\{a\}^+ \in (\mathbf{SPCF} \cap \mathbf{PPCF}^{-\varepsilon})$. □

Theorem 6.4.4. *$\mathbf{SPCF}$ and $\mathbf{0L}^{-\varepsilon}$ are incomparable, but not disjoint.*

Proof. Analogous to the proof of Theorem 6.4.3. □

Table 1. Mutual relations between investigated language families. A denotes the language family from the first column, B the language family from the table header. If the relation in the cell given by A and B is $\star$, then A $\star$ B. A$\|$B means that A and B are incomparable, but not disjoint, ? stands for an open problem, and the meaning of $\subset$, $=$, and $\supset$ is as usual.

A \ B	**SPCF**	**SPCF**$^{-\varepsilon}$	**JSPCF**	**JSPCF**$^{-\varepsilon}$	**PPCF**	**PPCF**$^{-\varepsilon}$	**JPPCF**	**JPPCF**$^{-\varepsilon}$	**0L**	**0L**$^{-\varepsilon}$
SPCF	$=$	$\supset$	$\parallel$	$\parallel$	$\subset$	$\parallel$	$\parallel$	$\parallel$	$\subset$	$\parallel$
SPCF$^{-\varepsilon}$	$\subset$	$=$	$\parallel$	$\parallel$	$\subset$	$\subset$	$\parallel$	$\parallel$	$\subset$	$\subset$
JSPCF	$\parallel$	$\parallel$	$=$	$\supset$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
JSPCF$^{-\varepsilon}$	$\parallel$	$\parallel$	$\subset$	$=$	$\parallel$	$\parallel$	?	?	$\parallel$	$\parallel$
PPCF	$\supset$	$\supset$	$\parallel$	$\parallel$	$=$	$\supset$	$\parallel$	$\parallel$	$\supset$	$\supset$
PPCF$^{-\varepsilon}$	$\parallel$	$\supset$	$\parallel$	$\parallel$	$\subset$	$=$	$\parallel$	$\parallel$	$\parallel$	$\supset$
JPPCF	$\parallel$	$\parallel$	$\parallel$	?	$\parallel$	$\parallel$	$=$	$\supset$	$\parallel$	$\parallel$
JPPCF$^{-\varepsilon}$	$\parallel$	$\parallel$	$\parallel$	?	$\parallel$	$\parallel$	$\subset$	$=$	$\parallel$	$\parallel$
0L	$\supset$	$\supset$	$\parallel$	$\parallel$	$\subset$	$\parallel$	$\parallel$	$\parallel$	$=$	$\supset$
0L$^{-\varepsilon}$	$\parallel$	$\supset$	$\parallel$	$\parallel$	$\subset$	$\subset$	$\parallel$	$\parallel$	$\subset$	$=$

The mutual relation between **JSPCF**$^{-\varepsilon}$ and **JPPCF**$^{-\varepsilon}$ is either incomparability or **JSPCF**$^{-\varepsilon}$ $\subset$ **JPPCF**$^{-\varepsilon}$, but the answer is not known as of now. The same goes for **JSPCF**$^{-\varepsilon}$ and **JPPCF**.

Open Problem 6.4.5. What is the relation between **JSPCF**$^{-\varepsilon}$ and **JPPCF**$^{-\varepsilon}$?

Open Problem 6.4.6. What is the relation between **JSPCF**$^{-\varepsilon}$ and **JPPCF**?

Theorem 6.4.7. PPCF$^{-\varepsilon}$ *and* **0L** *are incomparable, but not disjoint.*

Proof. Let $L_1 = \{aa, aab\}$ and $L_2 = \{a, aab\}$ be two languages over $\{a, b\}$. $L_1 \notin$ **PPCF**$^{-\varepsilon}$, $L_1 \in$ **0L**, $L_2 \in$ **PPCF**$^{-\varepsilon}$, and $L_2 \notin$ **0L** proves the incomparability, while $\{a\}^+ \in$ (**PPCF**$^{-\varepsilon} \cap$ **0L**) proves the disjointness. $\square$

In Table 1, a complete survery of all mutual relations between investigated language families is given. For completeness, the relation between **0L** and **0L**$^{-\varepsilon}$ (see [30]) is included as well.

If only unary alphabets are considered, the relations between investigated language families change rapidly. By Theorem 6.2.3, Corollary 6.2.9, and Corollary 6.2.10, for every unary alphabet the following holds

$$\mathbf{SPCF} = \mathbf{JSPCF} \subset \mathbf{PPCF} = \mathbf{JPPCF} = \mathbf{0L}$$

Trivially

$$\mathbf{SPCF}^{-\varepsilon} = \mathbf{JSPCF}^{-\varepsilon} \subset \mathbf{PPCF}^{-\varepsilon} = \mathbf{JPPCF}^{-\varepsilon} = \mathbf{0L}^{-\varepsilon}$$

$$\begin{tikzcd}
\mathbf{PPCF}^{-\varepsilon} \arrow[r, dash] \arrow[d] & \mathbf{JPPCF}^{-\varepsilon} \arrow[r, dash] \arrow[d] & \mathbf{0L}^{-\varepsilon} \arrow[d] \\
\mathbf{PPCF} \arrow[r, dash] & \mathbf{JPPCF} \arrow[r, dash] & \mathbf{0L} \\
\mathbf{SPCF} \arrow[u] \arrow[r, dash] & \mathbf{JSPCF} \arrow[u] & \\
\mathbf{SPCF}^{-\varepsilon} \arrow[u] \arrow[uuu, bend right] \arrow[r, dash] & \mathbf{JSPCF}^{-\varepsilon} \arrow[u] \arrow[uuu, bend right] &
\end{tikzcd}$$

Figure 2. Mutual relations between investigated language families in the case of unary alphabets. Straight line between two families means that these families are identical. The arrow from family A to family B denotes that A $\subset$ B.

holds for every unary alphabet as well. The following theorem demonstrates that for unary alphabets, **SPCF** and $\mathbf{PPCF}^{-\varepsilon}$ are incomparable, but not disjoint. Then, in Figure 2 the mutual relations between investigated language families for unary alphabets are visualized.

Theorem 6.4.8. *In case of unary alphabets,* **SPCF** *and* $\mathbf{PPCF}^{-\varepsilon}$ *are incomparable, but not disjoint.*

Proof. Clearly, the language $\{a\}^+$ is contained in both **SPCF** and $\mathbf{PPCF}^{-\varepsilon}$. Since the language $\{\varepsilon, a, aa\}$ from **SPCF** is not contained in $\mathbf{PPCF}^{-\varepsilon}$, $\mathbf{SPCF} \not\subseteq \mathbf{PPCF}^{-\varepsilon}$. Conversely, $\mathbf{PPCF}^{-\varepsilon} \not\subseteq \mathbf{SPCF}$ since $\mathbf{PPCF}^{-\varepsilon}$ is not semilinear. $\square$

Chapter 7
k#$-Rewriting Systems

As recalled in Chapter 3, #-rewriting systems of index k coincide with programmed grammars of the same index. This is because the number of nonterminal symbols that can simultaneously occur in a sentential form is limited by k in both language models. The fact that #-rewriting systems have only one nonterminal symbol, #, poses no problem here since they also use states and thus every combination of at most k nonterminal symbols can be simply encoded by a state (see proof in [52]).

This chapter presents a modification of #-rewriting systems of index k, called *k#$ rewriting systems*. The essence of the modification consists in extending #-rewriting systems with additional pushdown-like storage. More precisely, a configuration of a k#$-rewriting system is consist of two parts delimited by $. The part on the left is a configuration of a #-rewriting system, the part on the right is a pushdown-like storage. When the number of #s is going to exceed k during a rewriting step, the extra #s are transformed to nonterminal symbols and pushed to the pushdown-like storage. Conversely, when the number of #s is insufficient to perform a rewriting step, the missing #s are obtained by popping nonterminal symbols from the pushdown-like storage and transforming them back into #s. The information on how to transform a nonterminal symbol to # and back is recorded in a state.

The content of this chapter, published also in [48], is divided into two sections. Section 7.1 gives the definition of k#$-rewriting systems and demonstrates them on an example. Section 7.2 states and proves the theorems about the generative capacity of k#$-rewriting systems. It will be shown that k#$-rewriting systems have the same generative capacity as state grammars working in k-limited way. Furthermore, k#$-rewriting systems will be used as a tool to prove the proper inclusion of the language family generated by programmed grammars of index k in the language family generated by state grammars in a k-limited way.

7.1 Definition and Example

A formal definition of k#$-rewriting systems—which were roughly described at the beginning of this chapter—will be given in the following paragraphs.

Definition 7.1.1. Let $k \in \mathbb{N}$. A *k#$-rewriting system* (abbreviated k#$RS), M, is a quintuple $M = (Q, V, \Sigma, s, R)$, where

- Q is a finite set of *states*;
- V is a total alphabet, $V \cap Q = \emptyset$;
- Σ is an alphabet containing # and \$ called *bounders*, $\Sigma \subseteq V$;
- $s \in Q$ is the *initial state*;
- $R = (R_e \cup R_r \cup R_l)$, where

$$\begin{array}{rcl} R_e & \subseteq & (Q \times \mathbb{N} \times \{\#\} \times Q \times (\Sigma - \{\$\})^*) \\ R_r & \subseteq & (Q \times \{\#\} \times \{\$\} \times Q \times \{\$\} \times (V - \{\#, \$\})^*) \\ R_l & \subseteq & (Q \times \{\$\} \times (V - \Sigma) \times Q \times \{\#\} \times \{\$\}) \end{array}$$

  is a finite set of *rules*.

Rules $(p, n, \#, q, x) \in R$, $(p, \#, \$, q, \$, y) \in R$, and $(p, \$, A, q, \#, \$) \in R$, are usually written as $p\,{}_n\# \rightarrow qx \in R$, $p\#\$ \rightarrow q\$y \in R$, and $p\$A \rightarrow q\#\$ \in R$, respectively.

Let $\Xi \subseteq Q(\Sigma - \{\$\})^*\{\$\}(V - \{\#, \$\})^*$ such that $\chi \in \Xi$ if and only if $\mathscr{O}_\#(\chi) \leq k$. Then Ξ is the set of all *configurations* of M.

The M-based relation of *direct rewriting step*, $\Rightarrow_M$, over Ξ is defined as follows:

- $pu\#v\$\alpha \Rightarrow_M quxv\$\alpha$ if and only if $p\,{}_n\# \rightarrow qx \in R$, $\mathscr{O}_\#(u) = n - 1$, $p, q \in Q$, $u, v, x \in (\Sigma - \{\$\})^*$, $\alpha \in (V - \{\#, \$\})^*$, and $n \in \mathbb{N}$;
- $pu\#\$\alpha \Rightarrow_M qu\$x\alpha$ if and only if $p\#\$ \rightarrow q\$x \in R$, $p, q \in Q$, $u \in (\Sigma - \{\$\})^*$, and $x, \alpha \in (V - \{\#, \$\})^*$;
- $pu\$A\alpha \Rightarrow_M qu\#\$\alpha$ if and only if $p\$A \rightarrow q\#\$ \in R$, $p, q \in Q$, $u \in (\Sigma - \{\$\})^*$, $A \in (V - \Sigma)$, and $\alpha \in (V - \{\#, \$\})^*$;
- $pux\$\alpha \Rightarrow_M pu\$x\alpha$ if and only if $p \in Q$, $u \in (\Sigma - \{\$\})^*$, $x \in (\Sigma - \{\#, \$\})^*$, and $\alpha \in (V - \{\#, \$\})^*$;
- $pu\$x\alpha \Rightarrow_M pux\$\alpha$ if and only if $p \in Q$, $u \in (\Sigma - \{\$\})^*$, $x \in (\Sigma - \{\#, \$\})^*$, and $\alpha \in (V - \{\#, \$\})^*$.

For $m \geq 0$, $\Rightarrow_M^m$, $\Rightarrow_M^+$, and $\Rightarrow_M^*$ are defined as usual.

The language generated by M, L(M), is defined as

$$L(M) = \{w \mid s\#\$ \Rightarrow_M^* qw\$, q \in Q, w \in (\Sigma - \{\#, \$\})^*\}$$

The family of languages generated by k#\$-rewriting systems is denoted as $\mathscr{L}_k(\#\$RS)$. □

The following example demonstrates the generative capacity of k#\$-rewriting systems.

Example 7.1.2. Let $M = (Q, V, \Sigma, s, R)$ be a 2#\$-rewriting system, where

$$\begin{array}{rcl} Q & = & \{s, p, p', p^{(1)}, p^{(2)}, p^{(X)}, p^{(Y)}, q, f, f^{(A)}, f^{(B)}\} \\ V & = & \{A, B, X, a, b, c, d, 0, 1, \bar{0}, \bar{1}, [_1, [_2,]_1,]_2, \#, \$\} \\ \Sigma & = & \{a, b, c, d, 0, 1, \bar{0}, \bar{1}, [_1, [_2,]_1,]_2, \#, \$\} \end{array}$$

and R contains rules

$$
\begin{array}{rlrl}
1: & s\,{}_1\# \rightarrow p\#\# & 9: & p^{(\mathrm{Y})}\,{}_1\# \rightarrow q \\
2: & p\,{}_1\# \rightarrow p'a\#b & 10: & q\,{}_1\# \rightarrow f \\
3: & p'\,{}_2\# \rightarrow p^{(1)}c\# & 11: & f\$\mathrm{A} \rightarrow f^{(\mathrm{A})}\#\$ \\
4: & p'\,{}_2\# \rightarrow p^{(2)}d\# & 12: & f\$\mathrm{B} \rightarrow f^{(\mathrm{B})}\#\$ \\
5: & p^{(1)}\#\$ \rightarrow p^{(\mathrm{X})}\$\mathrm{X}[_1\mathrm{A}]_1 & 13: & f^{(\mathrm{A})}\,{}_1\# \rightarrow f^{(\mathrm{A})}0\#1 \\
6: & p^{(2)}\#\$ \rightarrow p^{(\mathrm{X})}\$\mathrm{X}[_2\mathrm{B}]_2 & 14: & f^{(\mathrm{B})}\,{}_1\# \rightarrow f^{(\mathrm{B})}\bar{0}\#\bar{1} \\
7: & p^{(\mathrm{X})}\$\mathrm{X} \rightarrow p\#\$ & 15: & f^{(\mathrm{A})}\,{}_1\# \rightarrow f01 \\
8: & p^{(\mathrm{X})}\$\mathrm{X} \rightarrow p^{(\mathrm{Y})}\#\$ & 16: & f^{(\mathrm{B})}\,{}_1\# \rightarrow f\bar{0}\bar{1}
\end{array}
$$

First, M generates two # bounders. Second, M uses rules 2 through 7 to generate the following structure

$$a^m\#b^m z_1 z_2 \cdots z_m\#\$\phi(z_m z_{m-1} \cdots z_1)$$

where $z_i \in \{c, d\}$, for all $1 \leq i \leq m$, $m \geq 1$, and ϕ is a morphism from $\{c, d\}^*$ to $\{\mathrm{A}, \mathrm{B}, [_1, [_2,]_1,]_2\}^*$ such that $\phi(c) = [_1\mathrm{A}]_1$ and $\phi(d) = [_2\mathrm{B}]_2$. Finally, M uses rules 8 through 16 to finish the rewriting. Thus, the language generated by M is

$$\mathrm{L(M)} = \left\{ w \;\middle|\; \begin{array}{l} w = a^n b^n z_1 z_2 \ldots z_n h(z_n, i_1) h(z_{n-1}, i_2) \ldots h(z_1, i_n) \\ z_i \in \{c, d\}, 1 \leq i \leq n, i_j \geq 1, 1 \leq j \leq n, n \geq 1 \end{array} \right\}$$

where h is a mapping from $\{c, d\} \times \mathbb{N}$ to $\{0, 1, \bar{0}, \bar{1}, [_1, [_2,]_1,]_2\}^*$ such that $h(c, i) = [_1 0^i 1^i]_1$ and $h(d, i) = [_2 \bar{0}^i \bar{1}^i]_2$.

For instance, M generates $aabbdc[_1 0011]_1[_2\bar{0}\bar{1}]_2$ in the following way

$$
\begin{array}{rlll}
s\#\$ & \Rightarrow_{\mathrm{M}} & p\#\#\$ & [1] \\
& \Rightarrow_{\mathrm{M}} & p'a\#b\#\$ & [2] \\
& \Rightarrow_{\mathrm{M}} & p^{(2)}a\#bd\#\$ & [4] \\
& \Rightarrow_{\mathrm{M}} & p^{(\mathrm{X})}a\#bd\$\mathrm{X}[_2\mathrm{B}]_2 & [6] \\
& \Rightarrow_{\mathrm{M}} & pa\#bd\#\$[_2\mathrm{B}]_2 & [7] \\
& \Rightarrow_{\mathrm{M}} & p'aa\#bbd\#\$[_2\mathrm{B}]_2 & [2] \\
& \Rightarrow_{\mathrm{M}} & p^{(1)}aa\#bbdc\#\$[_2\mathrm{B}]_2 & [3] \\
& \Rightarrow_{\mathrm{M}} & p^{(\mathrm{X})}aa\#bbdc\$\mathrm{X}[_1\mathrm{A}]_1[_2\mathrm{B}]_2 & [5] \\
& \Rightarrow_{\mathrm{M}} & p^{(\mathrm{Y})}aa\#bbdc\#\$[_1\mathrm{A}]_1[_2\mathrm{B}]_2 & [8] \\
& \Rightarrow_{\mathrm{M}} & qaabbdc\#\$[_1\mathrm{A}]_1[_2\mathrm{B}]_2 & [9] \\
& \Rightarrow_{\mathrm{M}} & faabbdc\$[_1\mathrm{A}]_1[_2\mathrm{B}]_2 & [10] \\
& \Rightarrow_{\mathrm{M}} & faabbdc[_1\$\mathrm{A}]_1[_2\mathrm{B}]_2 & \\
& \Rightarrow_{\mathrm{M}} & f^{(\mathrm{A})}aabbdc[_1\#\$]_1[_2\mathrm{B}]_2 & [11] \\
& \Rightarrow_{\mathrm{M}} & f^{(\mathrm{A})}aabbdc[_1 0\#1\$]_1[_2\mathrm{B}]_2 & [13] \\
& \Rightarrow_{\mathrm{M}} & faabbdc[_1 0011\$]_1[_2\mathrm{B}]_2 & [15] \\
& \Rightarrow_{\mathrm{M}} & faabbdc[_1 0011]_1[_2\$\mathrm{B}]_2 & \\
& \Rightarrow_{\mathrm{M}} & f^{(\mathrm{B})}aabbdc[_1 0011]_1[_2\#\$]_2 & [12] \\
& \Rightarrow_{\mathrm{M}} & faabbdc[_1 0011]_1[_2\bar{0}\bar{1}\$]_2 & [16] \\
& \Rightarrow_{\mathrm{M}} & faabbdc[_1 0011]_1[_2\bar{0}\bar{1}]_2\$ &
\end{array}
$$

□

7.2 Generative Capacity

First, the identity of $\mathbf{ST}_k$ and $\mathscr{L}_k(\#\$\mathrm{RS})$ for every $k \geq 1$ will be proven.

Lemma 7.2.1. *Let $k \geq 1$. Then,* $\mathbf{ST}_k \subseteq \mathscr{L}_k(\#\$\mathrm{RS})$.

Proof. Let $\mathrm{G} = (\mathrm{V}, \mathrm{T}, \mathrm{K}, \mathrm{P}, \mathrm{S}, s)$ be a state grammar. Without any loss on generality, suppose that $\mathrm{V} \cap \{\#, \$\} = \emptyset$. Now, construct a $k\#\$$-rewriting system

$$\mathrm{M} = (\mathrm{Q}, \mathrm{V}', \Sigma, s', \mathrm{R})$$

from G such that $\mathrm{L}(\mathrm{G}, k) = \mathrm{L}(\mathrm{M})$. First, set

$$\begin{array}{rcl} \mathrm{Q} & = & \bigcup_{i=0}^{k}\{\langle q; l; u\rangle \mid q \in \mathrm{K}, u \in (\mathrm{V} - \mathrm{T})^i, 0 \leq l \leq k\} \\ \mathrm{V}' & = & \mathrm{V} \cup \{\#, \$\} \\ \Sigma & = & \mathrm{T} \cup \{\#, \$\} \\ s' & = & \langle s; 0; \mathrm{S}\rangle \end{array}$$

In Q, each state contains three pieces of information—(1) the current state of G, (2) auxiliary substate of the simulation (0 stands for regular state, and 1 through k stand for auxiliary substates), and (3) the first k nonterminal symbols from the current sentential form of G. The positions of these k symbols correspond to #s in the simulation of G by M.

Next, construct R. Let

$$\mathrm{rules}(p, u) = \left\{ r \;\middle|\; \begin{array}{l} r\colon (\mathrm{B}, p) \to (x, q) \in \mathrm{P}, \mathrm{B} \in ((\mathrm{V} - \mathrm{T}) \cap \mathrm{alph}(u)) \\ p, q \in \mathrm{K}, x \in \mathrm{V}^+, u \in \mathrm{V}^* \end{array} \right\}$$

and let g and h be two morphisms from V^* to $(\Sigma - \{\$\})^*$ and from V^* to $(\mathrm{V}' - \Sigma)^*$, respectively, defined as

$$g(x) = \begin{cases} \# & \text{for every } x \in (\mathrm{V} - \mathrm{T}) \\ x & \text{for every } x \in \mathrm{T} \end{cases}$$

$$h(x) = \begin{cases} x & \text{for every } x \in (\mathrm{V} - \mathrm{T}) \\ \varepsilon & \text{for every } x \in \mathrm{T} \end{cases}$$

Initially, set $\mathrm{R} = \emptyset$. For every rule $(\mathrm{A}, p) \to (x, q) \in \mathrm{P}$ and for every state $\langle p; 0; u\mathrm{A}v\rangle \in \mathrm{Q}$ such that $\mathrm{rules}(p, u) = \emptyset$ perform the following steps:

(A) If $k - |uv| \geq |h(x)|$, then add $\langle p; 0; u\mathrm{A}v\rangle\ _{|u\mathrm{A}|}\# \to \langle q; 0; uh(x)v\rangle g(x)$ to R.

(B) If $k - |uv| < |h(x)|$, then express v as $v = \mathrm{X}_{m-1}\mathrm{X}_{m-2}\cdots\mathrm{X}_0$, where $\mathrm{X}_i \in (\mathrm{V}' - \Sigma)$, $0 \leq i \leq m - 1$, $m = |v|$, and

  (i) for every i such that $0 \leq i \leq m - 1$, add $\langle p; i; u\mathrm{A}v\rangle\#\$ \to \langle p; i + 1; u\mathrm{A}v\rangle\$\mathrm{X}_i$ to R;

(ii) add $\langle p; m; u\mathrm{A}v\rangle\#\$ \rightarrow \langle q; 0; u\rangle\$x$ to R.

Finally, for every state $\langle p; 0; u\rangle \in \mathrm{Q}$ such that $|u| \leq k - 1$ and for every $\mathrm{B} \in (\mathrm{V}' - \Sigma)$ add rule

$$\langle p; 0; u\rangle\$\mathrm{B} \rightarrow \langle p; 0; u\mathrm{B}\rangle\#\$$$

to R. The construction of M is completed.

Claim 7.2.2. *Let* $(\mathrm{S}, p)\ {}_k\!\Rightarrow^m_{\mathrm{G}} (wy, q)$*, where* $p, q \in \mathrm{K}$*,* $w \in \mathrm{T}^*$*,* $y \in ((\mathrm{V} - \mathrm{T})\mathrm{T}^*)^*$*, and* $m \geq 0$*. Then,* $\langle p; 0; \mathrm{S}\rangle\#\$ \Rightarrow^*_{\mathrm{M}} \langle q; 0; \mathrm{kprefix}(h(y), k)\rangle wg(\alpha)\$\beta$*, where* $y = \alpha\beta$*,* $\alpha \in (\mathrm{T}^*(\mathrm{V} - \mathrm{T}))^{|\mathrm{kprefix}(h(y),k)|}$*, and* $\beta \in \mathrm{V}^*$*.*

Proof. This claim is proved by induction on $m \geq 0$.

Basis. Let $m = 0$, so $(\mathrm{S}, p)\ {}_k\!\Rightarrow^0_{\mathrm{G}} (\mathrm{S}, p)$, $w = \varepsilon$ and $y = \mathrm{S}$. Then,

$$\langle p; 0; \mathrm{S}\rangle\#\$ \Rightarrow^0_{\mathrm{M}} \langle p; 0; \mathrm{kprefix}(h(\mathrm{S}), k)\rangle g(\alpha)\$\beta$$

Since $\mathrm{kprefix}(h(\mathrm{S}), k) = \mathrm{S}$, it holds that $\alpha = \mathrm{S}$ and $\beta = \varepsilon$, and hence

$$\langle p; 0; \mathrm{kprefix}(h(\mathrm{S}), k)\rangle g(\alpha)\$\beta = \langle p; 0; \mathrm{S}\rangle\#\$$$

Therefore, the basis holds.

Induction Hypothesis. Suppose that the claim holds for all $0 \leq m \leq l$, where l is a non-negative integer.

Induction Step. Let $(\mathrm{S}, p)\ {}_k\!\Rightarrow^{l+1}_{\mathrm{G}} (wy, q)$, where $p, q \in \mathrm{K}$, $w \in \mathrm{T}^*$, and $y \in ((\mathrm{V} - \mathrm{T})\mathrm{T}^*)^*$. Since $l + 1 \geq 1$, express $(\mathrm{S}, p)\ {}_k\!\Rightarrow^{l+1}_{\mathrm{G}} (wy, q)$ as $(\mathrm{S}, p)\ {}_k\!\Rightarrow^{l}_{\mathrm{G}} (w'u\mathrm{A}v, t)\ {}_k\!\Rightarrow_{\mathrm{G}} (w'uxv, q)$, where $t \in \mathrm{K}$, $w' \in \mathrm{T}^*$, $u \in ((\mathrm{V} - \mathrm{T})\mathrm{T}^*)^*$, $|h(u)| \leq k - 1$, $\mathrm{A} \in (\mathrm{V} - \mathrm{T})$, $x, v \in \mathrm{V}^*$, $(\mathrm{A}, t) \rightarrow (x, q) \in \mathrm{P}$, $w = w'\hat{w}$, and $\hat{w}y = uxv$ with $\hat{w} \in \mathrm{T}^*$. By the induction hypothesis, $\langle p; 0; \mathrm{S}\rangle\#\$ \Rightarrow^*_{\mathrm{M}} \langle t; 0; \mathrm{kprefix}(h(u\mathrm{A}v), k)\rangle w'g(u\mathrm{A}z)\$\bar{z}$, where $v = z\bar{z}$, $z \in (\mathrm{T}^*(\mathrm{V} - \mathrm{T}))^{|\mathrm{kprefix}(h(u\mathrm{A}v),k)| - |h(u\mathrm{A})|}$, and $\bar{z} \in \mathrm{V}^*$. As $(\mathrm{A}, t) \rightarrow (x, q) \in \mathrm{P}$, the following rules were added to R during its construction, based on the relation between $k - |\mathrm{kprefix}(h(u\mathrm{A}v), k)| + 1$ and $|h(x)|$:

(A) $k - |\mathrm{kprefix}(h(u\mathrm{A}v), k)| + 1 \geq |h(x)|$. Then, based on construction of R,

$$\langle t; 0; \mathrm{kprefix}(h(u\mathrm{A}v), k)\rangle\ {}_{|h(u\mathrm{A})|}\# \rightarrow \langle q; 0; \mathrm{kprefix}(h(uxv), k)\rangle g(x) \in \mathrm{R}$$

Now, the following three cases must be considered:

1. $\mathscr{O}_{\mathrm{V}-\mathrm{T}}(u\mathrm{A}v) \geq k$ and $\mathscr{O}_{\mathrm{V}-\mathrm{T}}(x) = 1$. Then,

$$\begin{aligned} &\langle t; 0; \mathrm{kprefix}(h(u\mathrm{A}v), k)\rangle w'g(u\mathrm{A}z)\$\bar{z} \\ \Rightarrow^*_{\mathrm{M}}\ &\langle q; 0; \mathrm{kprefix}(h(uxv), k)\rangle w'g(uxz)\$\bar{z} \end{aligned}$$

Clearly, uxz can be expressed as $\hat{w}\alpha$. As $h(uxv) = h(\hat{w}y) = h(y)$, $g(uxz) = g(\hat{w}\alpha) = \hat{w}g(\alpha)$, $w = w'\hat{w}$, and $\beta = \bar{z}$, the following holds

$$\begin{aligned} & \langle q; 0; \mathrm{kprefix}(h(uxv), k)\rangle w'g(uxz)\$\bar{z} \\ = \; & \langle q; 0; \mathrm{kprefix}(h(y), k)\rangle wg(\alpha)\$\beta \end{aligned}$$

2. $\mathscr{O}_{\mathrm{V}-\mathrm{T}}(u\mathrm{A}v) \geq k$ and $\mathscr{O}_{\mathrm{V}-\mathrm{T}}(x) = 0$. Then,

$$\begin{aligned} & \langle t; 0; \mathrm{kprefix}(h(u\mathrm{A}v), k)\rangle w'g(u\mathrm{A}z)\$\bar{z} \\ \Rightarrow^*_{\mathrm{M}} \; & \langle q; 0; h(uxz)\rangle w'g(uxz)\$\bar{z} \end{aligned}$$

and since for every $\mathrm{B} \in (\mathrm{V}' - \Sigma)$ it holds that

$$\langle q; 0; h(uxz)\rangle\$\mathrm{B} \to \langle q; 0; h(uxz\mathrm{B})\rangle\#\$ \in \mathrm{R}$$

there exists

$$\begin{aligned} & \langle q; 0; h(uxz)\rangle w'g(uxz)\$\bar{z} \\ \Rightarrow^*_{\mathrm{M}} \; & \langle q; 0; \mathrm{kprefix}(h(uxv), k)\rangle w'g(uxz')\$\bar{z}' \end{aligned}$$

with $z' \in (\mathrm{T}^*(\mathrm{V} - \mathrm{T}))^{|\mathrm{kprefix}(h(uxv),k)| - |h(ux)|}$, $\bar{z}' \in \mathrm{V}^*$, and $v = z'\bar{z}'$. Again, uxz' can be expressed as $\hat{w}\alpha$ and with $\beta = \bar{z}'$, the following holds

$$\begin{aligned} & \langle q; 0; \mathrm{kprefix}(h(uxv), k)\rangle w'g(uxz')\$\bar{z}' \\ = \; & \langle q; 0; \mathrm{kprefix}(h(y), k)\rangle wg(\alpha)\$\beta \end{aligned}$$

3. $\mathscr{O}_{\mathrm{V}-\mathrm{T}}(u\mathrm{A}v) < k$ and $\mathscr{O}_{\mathrm{V}-\mathrm{T}}(uxv) \leq k$. Then,

$$\begin{aligned} & \langle t; 0; \mathrm{kprefix}(h(u\mathrm{A}v), k)\rangle w'g(u\mathrm{A}z)\$\bar{z} \\ \Rightarrow^*_{\mathrm{M}} \; & \langle q; 0; \mathrm{kprefix}(h(uxv), k)\rangle w'g(uxz)\$\bar{z} \end{aligned}$$

As in previous cases, set $\hat{w}\alpha = uxz$ and $\beta = \bar{z}$. Therefore,

$$\begin{aligned} & \langle q; 0; \mathrm{kprefix}(h(uxv), k)\rangle w'g(uxz)\$\bar{z} \\ = \; & \langle q; 0; \mathrm{kprefix}(h(y), k)\rangle wg(\alpha)\$\beta \end{aligned}$$

(B) $k - |\mathrm{kprefix}(h(u\mathrm{A}v), k)| + 1 < |h(x)|$. Express $\mathrm{kprefix}(h(u\mathrm{A}v), k)$ as $h(u\mathrm{A})\delta$, where $\delta = \mathrm{D}_1\mathrm{D}_2 \ldots \mathrm{D}_{|\delta|}$, $\mathrm{D}_i \in (\mathrm{V} - \mathrm{T})$, $1 \leq i \leq |\delta|$. Furthermore, express z as $d_1\mathrm{D}_1d_2\mathrm{D}_2 \ldots d_{|\delta|}\mathrm{D}_{|\delta|}$, where $d_i \in \mathrm{T}^*$, $1 \leq i \leq |\delta|$. As there are the following rules in R introduced by step (B.i) of the construction of R

$$\begin{array}{rcl} \langle t; 0; h(u\mathrm{A})\delta\rangle\#\$ & \to & \langle t; 1; h(u\mathrm{A})\delta\rangle\$\mathrm{D}_{|\delta|} \\ \langle t; 1; h(u\mathrm{A})\delta\rangle\#\$ & \to & \langle t; 2; h(u\mathrm{A})\delta\rangle\$\mathrm{D}_{|\delta|-1} \\ & \vdots & \\ \langle t; |\delta| - 1; h(u\mathrm{A})\delta\rangle\#\$ & \to & \langle t; |\delta|; h(u\mathrm{A})\delta\rangle\$\mathrm{D}_1 \end{array}$$

there exists

$$\begin{aligned} & \langle t; 0; \mathrm{kprefix}(h(u\mathrm{A}v), k)\rangle w'g(u\mathrm{A}d_1\mathrm{D}_1 \ldots d_{|\delta|}\mathrm{D}_{|\delta|})\$\bar{z} \\ \Rightarrow^*_{\mathrm{M}} \; & \langle t; 1; h(u\mathrm{A})\delta\rangle w'g(u\mathrm{A}d_1\mathrm{D}_1 \ldots d_{|\delta|-1}\mathrm{D}_{|\delta|-1})\$d_{|\delta|}\mathrm{D}_{|\delta|}\bar{z} \\ \vdots \; & \\ \Rightarrow^*_{\mathrm{M}} \; & \langle t; |\delta|; h(u\mathrm{A})\delta\rangle w'g(u\mathrm{A})\$z\bar{z} \end{aligned}$$

Since step (B.ii) of the construction of R introduces a rule

$$\langle t; |\delta|; h(u\mathrm{A})\delta\rangle\#\$ \rightarrow \langle q; 0; h(u)\rangle\$x$$

to R, there also exists

$$\langle t; |\delta|; h(u\mathrm{A})\delta\rangle w'g(u\mathrm{A})\$z\bar{z} \Rightarrow^*_{\mathrm{M}} \langle q; 0; h(u)\rangle w'g(u)\$xz\bar{z}$$

Finally, for every state $\langle o; 0; \gamma\rangle \in \mathrm{Q}$ such that $|\gamma| \leq k-1$ and for every $\mathrm{B} \in (\mathrm{V}' - \Sigma)$, there is a rule

$$\langle o; 0; \gamma\rangle\$\mathrm{B} \rightarrow \langle o; 0; \gamma\mathrm{B}\rangle\#\$$$

in R and hence

$$\langle q; 0; h(u)\rangle w'g(u)\$xz\bar{z} \Rightarrow^*_{\mathrm{M}} \langle q; 0; \mathrm{kprefix}(h(uxv), k)\rangle w'g(uz')\$\bar{z}'$$

where $z' \in (\mathrm{T}^*(\mathrm{V}-\mathrm{T}))^{|\mathrm{kprefix}(h(uxv),k)|-|h(u)|}$, $\bar{z}' \in \mathrm{V}^*$, and $xv = z'\bar{z}'$. Express uz' as $\hat{w}\alpha$ and set $\bar{z}' = \beta$. Thus,

$$\langle q; 0; \mathrm{kprefix}(h(uxv), k)\rangle w'g(uz')\$\bar{z}' = \langle q; 0; \mathrm{kprefix}(h(y), k)\rangle wg(\alpha)\$\beta$$

and the claim holds. □

Claim 7.2.3. *Let* $\langle p; 0; \mathrm{S}\rangle\#\$ \Rightarrow^m_{\mathrm{M}} \langle q; i; h(\alpha\bar{\alpha})\rangle wg(\alpha)\$\bar{\alpha}\beta$, *where* $p, q \in \mathrm{K}$, $0 \leq i \leq k$, $w \in (\Sigma - \{\#, \$\})^*$, $\alpha \in ((\mathrm{V}' - \Sigma)(\mathrm{V}' - \{\#, \$\})^*)^*$, $\mathscr{O}_{\mathrm{V}'-\Sigma}(\alpha) \leq k$, $\bar{\alpha}, \beta \in (\mathrm{V}' - \{\#, \$\})^*$, $\mathscr{O}_{\mathrm{V}'-\Sigma}(\bar{\alpha}) = i$, *and* $m \geq 0$. *Then,* $(\mathrm{S}, p)\ {}_k\Rightarrow^*_{\mathrm{G}} (w\alpha\bar{\alpha}\beta, q)$.

Proof. This claim is proved by induction on $m \geq 0$.

Basis. Let $m = 0$, so $\langle p; 0; \mathrm{S}\rangle\#\$ \Rightarrow^0_{\mathrm{M}} \langle p; 0; \mathrm{S}\rangle\#\$$, $w = \varepsilon$, $\alpha = \mathrm{S}$, $\bar{\alpha} = \varepsilon$ and $\beta = \varepsilon$. Then, $(\mathrm{S}, p)\ {}_k\Rightarrow^*_{\mathrm{G}} (\mathrm{S}, p)$ and the basis holds.

Induction Hypothesis. Suppose that the claim holds for all $0 \leq m \leq l$, where l is a non-negative integer.

Induction Step. Let $\langle p; 0; \mathrm{S}\rangle\#\$ \Rightarrow^{l+1}_{\mathrm{M}} \langle q; \hat{j}; h(\alpha\bar{\alpha})\rangle wg(\alpha)\$\bar{\alpha}\beta$, where $p, q \in \mathrm{K}$, $0 \leq \hat{j} \leq k$, $w \in (\Sigma - \{\#, \$\})^*$, $\alpha \in ((\mathrm{V}' - \Sigma)(\mathrm{V}' - \{\#, \$\})^*)^*$, $\mathscr{O}_{\mathrm{V}'-\Sigma}(\alpha) \leq k$, and $\bar{\alpha}, \beta \in (\mathrm{V}' - \{\#, \$\})^*$, $\mathscr{O}_{\mathrm{V}'-\Sigma}(\bar{\alpha}) = \hat{j}$. Since $l + 1 \geq 1$, express $\langle p; 0; \mathrm{S}\rangle\#\$ \Rightarrow^{l+1}_{\mathrm{M}} \langle q; \hat{j}; h(\alpha\bar{\alpha})\rangle wg(\alpha)\$\bar{\alpha}\beta$ as

$$\langle p; 0; \mathrm{S}\rangle\#\$ \Rightarrow^l_{\mathrm{M}} \langle t; \hat{\imath}; h(u\mathrm{A}v\hat{\alpha})\rangle w'g(u\mathrm{A}v)\$\hat{\alpha}z \Rightarrow_{\mathrm{M}} \langle q; \hat{j}; h(uxv'\bar{\alpha})\rangle w'g(uxv')\$\bar{\alpha}\beta$$

where $t \in \mathrm{K}$, $0 \leq \hat{\imath} \leq k$, $w' \in (\Sigma - \{\#, \$\})^*$, $\mathrm{A} \in (\mathrm{V}' - \Sigma) \cup \{\varepsilon\}$, $u \in ((\mathrm{V}' - \Sigma)(\mathrm{V}' - \{\#, \$\})^*)^*$, $x, v, v', \hat{\alpha}, z \in (\mathrm{V}' - \{\#, \$\})^*$, $\mathscr{O}_{\mathrm{V}'-\Sigma}(u\mathrm{A}v) \leq k$, $\mathscr{O}_{\mathrm{V}'-\Sigma}(\hat{\alpha}) = \hat{\imath}$, $w = w'\hat{w}$, and $\hat{w}\alpha = uxv'$ with $\hat{w} \in (\Sigma - \{\#, \$\})^*$. By the induction hypothesis, $(\mathrm{S}, p)\ {}_k\Rightarrow^*_{\mathrm{G}} (w'u\mathrm{A}v\hat{\alpha}z, t)$. M can rewrite $\langle t; \hat{\imath}; h(u\mathrm{A}v\hat{\alpha})\rangle w'g(u\mathrm{A}v)\$\hat{\alpha}z$ to $\langle q; \hat{j}; h(uxv'\bar{\alpha})\rangle w'g(uxv')\$\bar{\alpha}\beta$ according to the one of following cases:

Case 1. $t = q$, $\hat{\imath} = \hat{j}$, $\mathrm{A} = x$, $v = v'\bar{v}$, $\bar{\alpha} = \bar{v}\hat{\alpha}$, $z = \beta$, and $\bar{v} \in (\Sigma - \{\#, \$\})^*$. In this case, the following holds

$$(w'u\mathrm{A}v'\bar{v}\hat{\alpha}z, t)\ {}_k\Rightarrow^*_{\mathrm{G}} (w\alpha\bar{\alpha}\beta, q)$$

Case 2. $t = q$, $\hat{\imath} = \hat{\jmath}$, $A = x$, $v' = v\bar{v}$, $\hat{\alpha} = \bar{v}\bar{\alpha}$, $z = \beta$, and $\bar{v} \in (\Sigma - \{\#, \$\})^*$. In this case, the following holds

$$(w'uAv\bar{v}\bar{\alpha}z, t) \; {}_k\!\Rightarrow^*_G \; (w\alpha\bar{\alpha}\beta, q)$$

Case 3. $t = q$, $\hat{\imath} = \hat{\jmath} = 0$, $A = x$, $\hat{\alpha} = \bar{\alpha} = \varepsilon$, $z = B\beta$, $v' = vB$, and $B \in (V' - \Sigma)$. In this case, the rewriting step was performed by rule $\langle q; 0; h(uv)\rangle\$B \to \langle q; 0; h(uvB)\rangle\#\$$, which was introduced to R for every state $\langle q; 0; h(uv)\rangle \in Q$, $|h(uv)| \leq k - 1$, for every $B \in (V' - \Sigma)$. Since this rule only changes symbol B to # and updates M's state to remember #'s meaning, the following holds

$$(w'uAv\hat{\alpha}B\beta, t) \; {}_k\!\Rightarrow^*_G \; (w'uxv'\bar{\alpha}\beta, q) \; {}_k\!\Rightarrow^*_G \; (w\alpha\bar{\alpha}\beta, q)$$

Case 4. $\hat{\imath} = \hat{\jmath} = 0$, $v = v'$, $\hat{\alpha} = \bar{\alpha}$ and $z = \beta$. In this case, $\langle t; 0; h(uAv)\rangle_{|h(uA)|}\# \to \langle q; 0; h(uxv)\rangle g(x) \in R$ was used, so there exists a rule $(A, t) \to (x, q)$ in P such that $\text{rules}(t, u) = \emptyset$ and hence

$$(w'uAv\hat{\alpha}z, t) \; {}_k\!\Rightarrow_G \; (w'uxv\hat{\alpha}z, q) \; {}_k\!\Rightarrow^*_G \; (w\alpha\bar{\alpha}\beta, q)$$

Case 5. $t = q$, $\hat{\imath} = \hat{\jmath} - 1$, $A = x$, $v = v'B$, $\bar{\alpha} = B\hat{\alpha}$, $z = \beta$, and $B \in (V' - \Sigma)$. In this case, $\langle t; \hat{\imath}; h(uAv\hat{\alpha})\rangle\#\$ \to \langle t; \hat{\imath}+1; h(uAv'\bar{\alpha})\rangle\$B \in R$ introduced in step (B.i) was used, and hence

$$(w'uAv'B\hat{\alpha}z, t) \; {}_k\!\Rightarrow^*_G \; (w\alpha\bar{\alpha}\beta, q)$$

Case 6. $\hat{\jmath} = 0$, $v = x = v' = \bar{\alpha} = \varepsilon$, $\beta = y\hat{\alpha}z$, and $y \in (V' - \{\#, \$\})^*$. In this case, $\langle t; \hat{\imath}; h(uA\hat{\alpha})\rangle\#\$ \to \langle q; 0; h(u)\rangle\$y \in R$ introduced in step (B.ii) was used, which means that there is a rule $(A, t) \to (y, q)$ in P such that $\text{rules}(t, u) = \emptyset$. Therefore,

$$(w'uAv\hat{\alpha}z, t) \; {}_k\!\Rightarrow_G \; (w'uyv\hat{\alpha}z, q) \; {}_k\!\Rightarrow^*_G \; (w\alpha\bar{\alpha}\beta, q)$$

which completes the induction step. □

If p and y in Claim 7.2.2 are set to s and ε, respectively, then $(S, s) \; {}_k\!\Rightarrow^*_G \; (w, q)$ implies $\langle s; 0; S\rangle\#\$ \Rightarrow^*_M \langle q; 0; \varepsilon\rangle w\$$ which proves $L(G, k) \subseteq L(M)$. Conversely, for $p = s$, $i = 0$, and $\alpha = \bar{\alpha} = \beta = \varepsilon$ in Claim 7.2.3, $\langle s; 0; S\rangle\#\$ \Rightarrow^*_M \langle q; 0; \varepsilon\rangle w\$$ implies $(S, s) \; {}_k\!\Rightarrow^*_G \; (w, q)$, which proves $L(M) \subseteq L(G, k)$. Hence, $L(G, k) = L(M)$ and the lemma holds. □

Lemma 7.2.4. *Let $k \geq 1$. Then, $\mathscr{L}_k(\#\$RS) \subseteq \mathbf{ST}_k$.*

Proof. Let $M = (Q, V, \Sigma, s, R)$ be a $k\#\$$-rewriting system. Without any loss on generality, suppose that ¿ $\notin V$ and $\#_i \notin V$, for all $1 \leq i \leq k$. From M, construct a state grammar

$$G = (V', T, K, P, \#_1, s')$$

such that $L(M) = L(G, k)$. First, set

$$\begin{array}{rcl}
V' & = & (V - \{\#, \$\}) \cup \{\#_i \mid 1 \leq i \leq k\} \\
T & = & \Sigma - \{\#, \$\} \\
K & = & \{\langle p; i\rangle \mid p \in Q, 0 \leq i \leq k\} \cup \{\langle p; i; r\rangle \mid p \in Q, 0 \leq i \leq k, r \in R\} \\
& \cup & \{\langle p; i; [\![r, j]\!]\rangle \mid p \in Q, r \in R, 0 \leq i \leq k, 1 \leq j \leq k\} \\
& \cup & \{\langle p; i; [\![r, X]\!]\rangle \mid p \in Q, r \in R, X \in (V - \Sigma) \cup \{¿\}, 0 \leq i \leq k\} \\
& \cup & \{q_{\text{fail}}\} \\
s' & = & \langle s; 1\rangle
\end{array}$$

In K, each state records the current state of M and the number of #s in the current configuration of M. Sometimes, it also contains the simulated rule, r, and additional information about either the leftmost non-# nonterminal symbol, X, or simulation progress, denoted by j. Note that X = ¿ if the simulation of a rule from R of the form r: $p\$\mathrm{A} \rightarrow q\#\$$ has started. In addition, K contains a special state q_{fail} that puts G to a configuration that rules out the next derivation step in G, which unsuccessfully stops the simulation of M.

Let τ be a mapping from $(\Sigma - \{\$\})^* \times \{1, 2, \dots, k\}$ to $(\mathrm{T} \cup \{\#_i \mid 1 \leq i \leq k\})^*$ defined recursively as follows

- $\tau(\varepsilon, i) = \varepsilon$, for every $1 \leq i \leq k$
- $\tau(ax, i) = a\tau(x, i)$, for every $a \in (\Sigma - \{\#, \$\})$, $x \in (\Sigma - \{\$\})^*$, and $1 \leq i \leq k$
- $\tau(\#x, i) = \#_i\tau(x, i+1)$, for every $x \in (\Sigma - \{\$\})^*$ and $1 \leq i \leq k-1$

Now, construct P. Initially, set P = $\emptyset$. For every state $\langle p; \kappa\rangle \in$ K and for every rule r: $p\,{}_n\# \rightarrow qx \in$ R such that $n \leq \kappa$ and $\kappa - 1 + \mathscr{O}_\#(x) \leq k$ perform the following steps:

(A) If $\mathscr{O}_\#(x) = 0$ and $\kappa - n = 0$, then add

$$\begin{aligned}(\#_1, \langle p; \kappa\rangle) &\rightarrow (\#_1, \langle p; \kappa; r\rangle)\\(\#_\kappa, \langle p; \kappa; r\rangle) &\rightarrow (x, \langle q; \kappa - 1\rangle)\end{aligned}$$

to P.

(B) If $\mathscr{O}_\#(x) = 0$ and $\kappa - n \geq 1$, then

- add $(\#_1, \langle p; \kappa\rangle) \rightarrow (\#_1, \langle p; \kappa; r\rangle)$ to P;
- add $(\#_n, \langle p; \kappa; r\rangle) \rightarrow (x, \langle q; \kappa - 1; [\![r, 1]\!]\rangle)$ to P;
- for every $1 \leq i \leq \kappa - n - 1$, add

$$(\#_{n+i}, \langle q; \kappa - 1; [\![r, i]\!]\rangle) \rightarrow (\#_{n+i-1}, \langle q; \kappa - 1; [\![r, i+1]\!]\rangle)$$

  to P;
- add $(\#_\kappa, \langle q; \kappa - 1; [\![r, \kappa - n]\!]\rangle) \rightarrow (\#_{\kappa-1}, \langle q; \kappa - 1\rangle)$ to P.

(C) If $\mathscr{O}_\#(x) = 1$, then add

$$\begin{aligned}(\#_1, \langle p; \kappa\rangle) &\rightarrow (\#_1, \langle p; \kappa; r\rangle)\\(\#_n, \langle p; \kappa; r\rangle) &\rightarrow (\tau(x, n), \langle q; \kappa\rangle)\end{aligned}$$

to P.

(D) If $\mathscr{O}_\#(x) \geq 2$, then

- add $(\#_1, \langle p; \kappa\rangle) \rightarrow (\#_1, \langle p; \kappa; r\rangle)$ to P;
- add $(\#_n, \langle p; \kappa; r\rangle) \rightarrow (\#_n, \langle p; \kappa; [\![r, 1]\!]\rangle)$ to P;
- for every $0 \leq i \leq \kappa - n - 1$, add

$$(\#_{\kappa-i}, \langle p; \kappa; [\![r, i+1]\!]\rangle) \rightarrow (\#_{\kappa+\eta-i}, \langle p; \kappa; [\![r, i+2]\!]\rangle)$$

to P, where $\eta = \mathscr{O}_{\#}(x) - 1$;

- add $(\#_n, \langle p; \kappa; [\![r, \kappa - n + 1]\!]\rangle) \rightarrow (\tau(x, n), \langle q; \kappa + \eta\rangle)$ to P, where $\eta = \mathscr{O}_{\#}(x) - 1$.

Next, for every state $\langle p; \kappa\rangle \in$ K such that $\kappa \geq 1$ and for every rule r: $p\#\$ \rightarrow q\$x \in$ R, add

$$\begin{array}{rcl}(\#_1, \langle p; \kappa\rangle) & \rightarrow & (\#_1, \langle p; \kappa; r\rangle) \\ (\#_\kappa, \langle p; \kappa; r\rangle) & \rightarrow & (x, \langle q; \kappa - 1\rangle)\end{array}$$

to P.

Finally, for every state $\langle p; \kappa\rangle \in$ K such that $\kappa \leq k-1$ and for every rule r: $p\$A \rightarrow q\#\$ \in$ R, add

- $(A, \langle p; \kappa\rangle) \rightarrow (A, \langle p; \kappa; r\rangle)$ if $\kappa = 0$
- $(\#_1, \langle p; \kappa\rangle) \rightarrow (\#_1, \langle p; \kappa; r\rangle)$ if $\kappa \geq 1$
- $(A, \langle p; \kappa; r\rangle) \rightarrow (A, \langle p; \kappa; [\![r, ¿]\!]\rangle)$
- $(X, \langle p; \kappa; [\![r, ¿]\!]\rangle) \rightarrow (X, \langle p; \kappa; [\![r, X]\!]\rangle)$, for all $X \in (V - \Sigma)$
- $(Y, \langle p; \kappa; [\![r, Y]\!]\rangle) \rightarrow (Y, q_{\text{false}})$, for all $Y \in (V - \Sigma)$, where $Y \neq A$
- $(A, \langle p; \kappa; [\![r, A]\!]\rangle) \rightarrow (\#_{\kappa+1}, \langle q; \kappa + 1\rangle)$

to P. Now, the construction of G is completed.

Claim 7.2.5. *Let $p\#\$ \Rightarrow^m_M qw\alpha\$\beta$, where $p, q \in Q$, $w \in (\Sigma - \{\#, \$\})^*$, $\alpha \in (\{\#\}(\Sigma - \{\$\})^*)^*$, $\beta \in (V - \{\#, \$\})^*$, and $m \geq 0$. Then,*

$$(\#_1, \langle p; 1\rangle) \;{}_k\Rightarrow^*_G (w\tau(\alpha, 1)\beta, \langle q; \mathscr{O}_{\#}(\alpha)\rangle)$$

Proof. This claim is proved by induction on $m \geq 0$.

Basis. Let $m = 0$, $p\#\$ \Rightarrow^0_M p\#\$$, $w = \varepsilon$, $\alpha = \#$, and $\beta = \varepsilon$. Then, $(\#_1, \langle p; 1\rangle) \;{}_k\Rightarrow^*_G (\#_1, \langle p; 1\rangle)$ and the basis holds.

Induction Hypothesis. Suppose that the claim holds for all $0 \leq m \leq l$, where l is a non-negative integer.

Induction Step. Let $p\#\$ \Rightarrow^{l+1}_M qw\alpha\$\beta$, where $p, q \in Q$, $w \in (\Sigma - \{\#, \$\})^*$, $\alpha \in (\{\#\}(\Sigma - \{\$\})^*)^*$, and $\beta \in (V - \{\#, \$\})^*$. Since $l + 1 \geq 1$, express $p\#\$ \Rightarrow^{l+1}_M qw\alpha\$\beta$ as

$$p\#\$ \Rightarrow^l_M tw'uA_{\#}v\$z \Rightarrow_M qw'uxv'\$\beta$$

where $t \in Q$, $w' \in (\Sigma - \{\#, \$\})^*$, $u \in (\{\#\}(\Sigma - \{\$\})^*)^*$, $A_{\#} \in \{\#, \varepsilon\}$, $x, v, v' \in (\Sigma - \{\$\})^*$, $z \in (V - \{\#, \$\})^*$, $w = w'\hat{w}$, and $\hat{w}\alpha = uxv'$ with $\hat{w} \in (\Sigma - \{\#, \$\})^*$. By the induction hypothesis, $(\#_1, \langle p; 1\rangle) \;{}_k\!\Rightarrow^*_{\mathrm{G}}\; (w'\tau(u\mathrm{A}_\# v, 1)z, \langle t; \mathscr{O}_\#(u\mathrm{A}_\# v)\rangle)$. M can perform $tw'u\mathrm{A}_\# v\$z \Rightarrow_{\mathrm{M}} qw'uxv'\$\beta$ in the following ways:

(I) $tw'u\mathrm{A}_\# v\$z \Rightarrow_{\mathrm{M}} qw'uxv'\$\beta$, where $t = q$, $\mathrm{A}_\# = x$, $v = v'\bar{v}$, $\beta = \bar{v}z$, and $\bar{v} \in (\Sigma - \{\#, \$\})^*$. This rewriting step only transfers terminal symbols from the left to the right relatively to \$. Clearly,

$$\begin{aligned} & (w'\tau(u\mathrm{A}_\# v, 1)z, \langle t; \mathscr{O}_\#(u\mathrm{A}_\# v)\rangle) \\ {}_k\!\Rightarrow^*_{\mathrm{G}}\; & (w'\tau(uxv', 1)\beta, \langle q; \mathscr{O}_\#(uxv')\rangle) \\ {}_k\!\Rightarrow^*_{\mathrm{G}}\; & (w\tau(\alpha, 1)\beta, \langle q; \mathscr{O}_\#(\alpha)\rangle) \end{aligned}$$

(II) $tw'u\mathrm{A}_\# v\$z \Rightarrow_{\mathrm{M}} qw'uxv'\$\beta$, where $t = q$, $\mathrm{A}_\# = x$, $v' = v\bar{v}$, $\bar{v} = \bar{v}\beta$, and $\bar{v} \in (\Sigma - \{\#, \$\})^*$. This rewriting step only transfers terminal symbols from the right to the left relatively to \$. Thus,

$$\begin{aligned} & (w'\tau(u\mathrm{A}_\# v, 1)z, \langle t; \mathscr{O}_\#(u\mathrm{A}_\# v)\rangle) \\ {}_k\!\Rightarrow^*_{\mathrm{G}}\; & (w'\tau(uxv', 1)\beta, \langle q; \mathscr{O}_\#(uxv')\rangle) \\ {}_k\!\Rightarrow^*_{\mathrm{G}}\; & (w\tau(\alpha, 1)\beta, \langle q; \mathscr{O}_\#(\alpha)\rangle) \end{aligned}$$

(III) $tw'u\mathrm{A}_\# v\$z \Rightarrow_{\mathrm{M}} qw'uxv'\$\beta$, where $\mathrm{A}_\# = \#$, $v = v'$, $z = \beta$, $\mathscr{O}_\#(u) = n - 1$, and $1 \le n \le k$. This rewriting step was performed by applying a rule r: $t\,{}_n\# \to qx \in \mathrm{R}$. Set $\kappa = \mathscr{O}_\#(u\mathrm{A}_\# v)$. G simulates application of r in one of the following ways:

(1) $\kappa - n = 0$ and $\mathscr{O}_\#(x) = 0$. Then, $(\#_1, \langle t; \kappa\rangle) \to (\#_1, \langle t; \kappa; r\rangle) \in \mathrm{P}$ and $(\#_\kappa, \langle t; \kappa; r\rangle) \to (x, \langle q; \kappa - 1\rangle) \in \mathrm{P}$, so

$$\begin{aligned} & (w'\tau(u\mathrm{A}_\# v, 1)z, \langle t; \mathscr{O}_\#(u\mathrm{A}_\# v)\rangle) \\ {}_k\!\Rightarrow_{\mathrm{G}}\; & (w'\tau(u\mathrm{A}_\# v, 1)z, \langle t; \mathscr{O}_\#(u\mathrm{A}_\# v); r\rangle) \\ {}_k\!\Rightarrow_{\mathrm{G}}\; & (w'\tau(uxv', 1)\beta, \langle q; \mathscr{O}_\#(uxv')\rangle) \\ {}_k\!\Rightarrow^*_{\mathrm{G}}\; & (w\tau(\alpha, 1)\beta, \langle q; \mathscr{O}_\#(\alpha)\rangle) \end{aligned}$$

where $\mathscr{O}_\#(u\mathrm{A}_\# v) - 1 = \kappa - 1 = \mathscr{O}_\#(uxv')$.

(2) $\kappa - n \ge 1$ and $\mathscr{O}_\#(x) = 0$. Then, $(\#_1, \langle t; \kappa\rangle) \to (\#_1, \langle t; \kappa; r\rangle) \in \mathrm{P}$ and $(\#_n, \langle t; \kappa; r\rangle) \to (x, \langle q; \kappa - 1; [\![r, 1]\!]\rangle) \in \mathrm{P}$, so

$$\begin{aligned} & (w'\tau(u\mathrm{A}_\# v, 1)z, \langle t; \mathscr{O}_\#(u\mathrm{A}_\# v)\rangle) \\ {}_k\!\Rightarrow_{\mathrm{G}}\; & (w'\tau(u\mathrm{A}_\# v, 1)z, \langle t; \mathscr{O}_\#(u\mathrm{A}_\# v); r\rangle) \\ {}_k\!\Rightarrow_{\mathrm{G}}\; & (w'\tau(ux, 1)\tau(v', n + 1)\beta, \langle q; \kappa - 1; [\![r, 1]\!]\rangle) \end{aligned}$$

If $\kappa - n \ge 2$, then there are rules

$$\begin{aligned} (\#_{n+1}, \langle q; \kappa - 1; [\![r, 1]\!]\rangle) &\to (\#_n, \langle q; \kappa - 1; [\![r, 2]\!]\rangle) \\ (\#_{n+2}, \langle q; \kappa - 1; [\![r, 2]\!]\rangle) &\to (\#_{n+1}, \langle q; \kappa - 1; [\![r, 3]\!]\rangle) \\ &\;\;\vdots \\ (\#_{\kappa-1}, \langle q; \kappa - 1; [\![r, \kappa - n - 1]\!]\rangle) &\to (\#_{\kappa-2}, \langle q; \kappa - 1; [\![r, \kappa - n]\!]\rangle) \end{aligned}$$

in P. Set $\bar{\eta} = \kappa - n = \mathscr{O}_\#(v')$. Since $\kappa - n \geq 2$, it follows that also $\bar{\eta} \geq 2$, so v' can be expressed as $v' = \delta_1\delta_2 \ldots \delta_{\bar{\eta}}$, where $\delta_i \in (\Sigma - \{\#, \$\})^*\{\#\}(\Sigma - \{\#, \$\})^*$, for all $1 \leq i \leq \bar{\eta}$, and G can perform the following sequence of derivation steps:

$$\begin{aligned}
& (w'\tau(ux, 1)\tau(\delta_1\delta_2 \ldots \delta_{\bar{\eta}}, n + 1)\beta, \langle q; \kappa - 1; [\![r, 1]\!]\rangle) \\
{}_k\!\Rightarrow_{\mathrm{G}} & (w'\tau(ux\delta_1, 1)\tau(\delta_2\delta_3 \ldots \delta_{\bar{\eta}}, n + 2)\beta, \langle q; \kappa - 1; [\![r, 2]\!]\rangle) \\
{}_k\!\Rightarrow_{\mathrm{G}} & (w'\tau(ux\delta_1\delta_2, 1)\tau(\delta_3\delta_4 \ldots \delta_{\bar{\eta}}, n + 3)\beta, \langle q; \kappa - 1; [\![r, 3]\!]\rangle) \\
\vdots & \\
{}_k\!\Rightarrow_{\mathrm{G}} & (w'\tau(ux\delta_1\delta_2 \ldots \delta_{\bar{\eta}-1}, 1)\tau(\delta_{\bar{\eta}}, n + \bar{\eta})\beta, \langle q; \kappa - 1; [\![r, \bar{\eta}]\!]\rangle)
\end{aligned}$$

The simulation of r is finished by the rule

$$(\#_\kappa, \langle q; \kappa - 1; [\![r, \kappa - n]\!]\rangle) \to (\#_{\kappa-1}, \langle q; \kappa - 1\rangle) \in \mathrm{P}$$

If $\kappa - n \geq 2$, then

$$\begin{aligned}
& (w'\tau(ux\delta_1\delta_2 \ldots \delta_{\bar{\eta}-1}, 1)\tau(\delta_{\bar{\eta}}, n + \bar{\eta})\beta, \langle q; \kappa - 1; [\![r, \bar{\eta}]\!]\rangle) \\
{}_k\!\Rightarrow_{\mathrm{G}} & (w'\tau(ux\delta_1\delta_2 \ldots \delta_{\bar{\eta}}, 1)\beta, \langle q; \kappa - 1\rangle) \\
{}_k\!\Rightarrow^*_{\mathrm{G}} & (w\tau(\alpha)\beta, \langle q; \mathscr{O}_\#(\alpha)\rangle)
\end{aligned}$$

Otherwise, $\kappa - n = 1$, and

$$\begin{aligned}
& (w'\tau(ux, 1)\tau(v', n + 1)\beta, \langle q; \kappa - 1; [\![r, 1]\!]\rangle) \\
{}_k\!\Rightarrow_{\mathrm{G}} & (w'\tau(uxv', 1)\beta, \langle q; \kappa - 1\rangle) \\
{}_k\!\Rightarrow^*_{\mathrm{G}} & (w\tau(\alpha)\beta, \langle q; \mathscr{O}_\#(\alpha)\rangle)
\end{aligned}$$

where $\tau(v', n + 1) = v_1\#_\kappa v_2$, $v_1, v_2 \in \mathrm{T}^*$, and $\mathscr{O}_\#(uxv') = \kappa - 1 = \mathscr{O}_\#(\alpha)$.

(3) $\mathscr{O}_\#(x) = 1$. Then, $(\#_1, \langle t; \kappa\rangle) \to (\#_1, \langle t; \kappa; r\rangle) \in \mathrm{P}$ and $(\#_n, \langle t; \kappa; r\rangle) \to (\tau(x, n), \langle q; \kappa\rangle) \in \mathrm{P}$, so

$$\begin{aligned}
& (w'\tau(u\mathrm{A}_\# v, 1)z, \langle t; \mathscr{O}_\#(u\mathrm{A}_\# v)\rangle) \\
{}_k\!\Rightarrow_{\mathrm{G}} & (w'\tau(u\mathrm{A}_\# v, 1)z, \langle t; \mathscr{O}_\#(u\mathrm{A}_\# v); r\rangle) \\
{}_k\!\Rightarrow_{\mathrm{G}} & (w'\tau(uxv, 1)z, \langle q; \mathscr{O}_\#(uxv)\rangle) \\
{}_k\!\Rightarrow^*_{\mathrm{G}} & (w\tau(\alpha, 1)\beta, \langle q; \mathscr{O}_\#(\alpha)\rangle)
\end{aligned}$$

(4) $\mathscr{O}_\#(x) \geq 2$. Set $\eta = \mathscr{O}_\#(x) - 1$. If $\kappa - n \geq 1$, then the following rules were introduced to P:

$$\begin{aligned}
(\#_1, \langle t; \kappa\rangle) &\to (\#_1, \langle t; \kappa; r\rangle) \\
(\#_n, \langle t; \kappa; r\rangle) &\to (\#_n, \langle t; \kappa; [\![r, 1]\!]\rangle) \\
(\#_\kappa, \langle t; \kappa; [\![r, 1]\!]\rangle) &\to (\#_{\kappa+\eta}, \langle t; \kappa; [\![r, 2]\!]\rangle) \\
(\#_{\kappa-1}, \langle t; \kappa; [\![r, 2]\!]\rangle) &\to (\#_{\kappa+\eta-1}, \langle t; \kappa; [\![r, 3]\!]\rangle) \\
&\vdots \\
(\#_{n+1}, \langle t; \kappa; [\![r, \kappa - n]\!]\rangle) &\to (\#_{\eta+n+1}, \langle t; \kappa; [\![r, \kappa - n + 1]\!]\rangle) \\
(\#_n, \langle t; \kappa; [\![r, \kappa - n + 1]\!]\rangle) &\to (\tau(x, n), \langle q; \kappa + \eta\rangle)
\end{aligned}$$

Set $\bar{\eta} = \kappa - n$ and express v as $v = \delta_1\delta_2 \ldots \delta_{\bar{\eta}}$, where

$$\delta_i \in (\Sigma - \{\#, \$\})^*\{\#\}(\Sigma - \{\#, \$\})^*$$

for all $1 \leq i \leq \bar{\eta}$. Then, G is able to perform the following sequence of derivation steps:

$$\begin{array}{ll} & (w'\tau(uA_{\#}v, 1)z, \langle t; \kappa\rangle) \\ {}_k\Rightarrow_G & (w'\tau(uA_{\#}v, 1)z, \langle t; \kappa; r\rangle) \\ {}_k\Rightarrow_G & (w'\tau(uA_{\#}\delta_1\delta_2 \dots \delta_{\bar{\eta}}, 1)z, \langle t; \kappa; [\![r, 1]\!]\rangle) \\ {}_k\Rightarrow_G & (w'\tau(uA_{\#}\delta_1\delta_2 \dots \delta_{\bar{\eta}-1}, 1)\tau(\delta_{\bar{\eta}}, \kappa + \eta)z, \langle t; \kappa; [\![r, 2]\!]\rangle) \\ {}_k\Rightarrow_G & (w'\tau(uA_{\#}\delta_1\delta_2 \dots \delta_{\bar{\eta}-2}, 1)\tau(\delta_{\bar{\eta}-1}\delta_{\bar{\eta}}, \kappa + \eta - 1)z, \langle t; \kappa; [\![r, 3]\!]\rangle) \\ \vdots & \\ {}_k\Rightarrow_G & (w'\tau(uA_{\#}, 1)\tau(\delta_1\delta_2 \dots \delta_{\bar{\eta}}, \eta + n + 1)z, \langle t; \kappa; [\![r, \kappa - n + 1]\!]\rangle) \\ {}_k\Rightarrow_G & (w'\tau(uxv, 1)z, \langle q; \kappa + \eta\rangle) \\ {}_k\Rightarrow_G^* & (w\tau(\alpha, 1)\beta, \langle q; \mathcal{O}_{\#}(\alpha)\rangle) \end{array}$$

Observe that $\kappa + \eta - (\kappa - n - 1) = \eta + n + 1$ and $\mathcal{O}_{\#}(uxv) = \kappa + \eta = \mathcal{O}_{\#}(\alpha)$ since r removes one and add $\mathcal{O}_{\#}(x)$ # symbols.
If $\kappa - n = 0$, only the rules

$$\begin{array}{rcl} (\#_1, \langle t; \kappa\rangle) & \to & (\#_1, \langle t; \kappa; r\rangle) \\ (\#_n, \langle t; \kappa; r\rangle) & \to & (\#_n, \langle t; \kappa; [\![r, 1]\!]\rangle) \\ (\#_n, \langle t; \kappa; [\![r, 1]\!]\rangle) & \to & (\tau(x, n), \langle q; \kappa + \eta\rangle) \end{array}$$

from P were used during the simulation of r by G as demonstrated by the following sequence of derivation steps:

$$\begin{array}{ll} & (w'\tau(uA_{\#}v, 1)z, \langle t; \kappa\rangle) \\ {}_k\Rightarrow_G & (w'\tau(uA_{\#}v, 1)z, \langle t; \kappa; r\rangle) \\ {}_k\Rightarrow_G & (w'\tau(uA_{\#}v, 1)z, \langle t; \kappa; [\![r, 1]\!]\rangle) \\ {}_k\Rightarrow_G & (w'\tau(uxv, 1)z, \langle q; \kappa + \eta\rangle) \\ {}_k\Rightarrow_G^* & (w\tau(\alpha, 1)\beta, \langle q; \mathcal{O}_{\#}(\alpha)\rangle) \end{array}$$

(IV) $tw'uA_{\#}v\$z \Rightarrow_M qw'uxv'\$\beta$, where $A_{\#} = \#$, $x = v = v' = \varepsilon$, $\beta = yz$, and $y \in (V - \{\#, \$\})^*$. Then, a rule r: $t\#\$ \to q\$y \in R$ was applied and hence $(\#_1, \langle t; \kappa\rangle) \to (\#_1, \langle t; \kappa; r\rangle) \in P$ and $(\#_\kappa, \langle t; \kappa; r\rangle) \to (x, \langle q; \kappa - 1\rangle) \in P$. Hence,

$$\begin{array}{ll} & (w'\tau(uA_{\#}v, 1)z, \langle t; \kappa\rangle) \\ {}_k\Rightarrow_G & (w'\tau(uA_{\#}v, 1)z, \langle t; \kappa; r\rangle) \\ {}_k\Rightarrow_G & (w'\tau(u, 1)yz, \langle q; \kappa - 1\rangle) \\ {}_k\Rightarrow_G^* & (w\tau(\alpha, 1)\beta, \langle q; \mathcal{O}_{\#}(\alpha)\rangle) \end{array}$$

(V) $tw'uA_{\#}v\$z \Rightarrow_M qw'uxv'\$\beta$, where $A_{\#} = \varepsilon$, $x = v = \varepsilon$, $v' = \#$, $z = A\beta$, and $A \in (V - \Sigma)$. In this case, a rule r: $t\$A \to q\#\$ \in R$ was applied, so for every $X, Y \in (V - \Sigma)$, where $Y \neq A$, the following rules

$$\begin{array}{rcl} (A, \langle t; \kappa\rangle) & \to & (A, \langle t; \kappa; r\rangle) \text{ if } \kappa = 0 \\ (\#_1, \langle t; \kappa\rangle) & \to & (\#_1, \langle t; \kappa; r\rangle) \text{ if } \kappa \geq 1 \\ (A, \langle t; \kappa; r\rangle) & \to & (A, \langle t; \kappa; [\![r, ¿]\!]\rangle) \\ (X, \langle t; \kappa; [\![r, ¿]\!]\rangle) & \to & (X, \langle t; \kappa; [\![r, X]\!]\rangle) \\ (Y, \langle t; \kappa; [\![r, Y]\!]\rangle) & \to & (Y, q_{\text{false}}) \\ (A, \langle t; \kappa; [\![r, A]\!]\rangle) & \to & (\#_{\kappa+1}, \langle q; \kappa + 1\rangle) \end{array}$$

were introduced in P. Hence, G simulates the application of r in the following way:

$$\begin{array}{rl} & (w'\tau(uA_{\#}v, 1)A\beta, \langle t; \kappa\rangle) \\ {}_k\Rightarrow_G & (w'\tau(uA_{\#}v, 1)A\beta, \langle t; \kappa; r\rangle) \\ {}_k\Rightarrow_G & (w'\tau(uA_{\#}v, 1)A\beta, \langle t; \kappa; [\![r, ¿]\!]\rangle) \\ {}_k\Rightarrow_G & (w'\tau(uA_{\#}v, 1)A\beta, \langle t; \kappa; [\![r, A]\!]\rangle) \\ {}_k\Rightarrow_G & (w'\tau(ux\#, 1)\beta, \langle q; \kappa + 1\rangle) \\ {}_k\Rightarrow^*_G & (w\tau(\alpha, 1)\beta, \langle q; \mathscr{O}_{\#}(\alpha)\rangle) \end{array}$$

Observe that A must be the first nonterminal symbol just behind $\#_i$ nonterminals, $1 \le i \le k$. When $z = BA\beta$, with $B \in (V - \Sigma)$ and $B \neq A$, then G reaches the state q_{false} and the simulation is blocked. □

Claim 7.2.6. *Set* $\Omega = \{[\![r, X]\!] \mid r \in R, X \in (\{1, 2, \dots, k\} \cup (V - \Sigma) \cup \{¿\})\} \cup R$ *and express* K *as* $K = K_Q \cup K_\Omega \cup \{q_{\text{false}}\}$, *where*

$$\begin{array}{rcl} K_Q & = & \{\langle p; i\rangle \mid p \in Q, 0 \le i \le k\} \\ K_\Omega & = & \{\langle p; i; Z\rangle \mid p \in Q, 0 \le i \le k, Z \in \Omega\} \end{array}$$

Define a binary operation • *from* $K_Q \times (\Omega \cup \{\lambda\})$ *to* K *such that*

$$\begin{array}{rcll} \langle p; i\rangle \bullet Z & = & \langle p; i; Z\rangle, & \textit{for all } Z \in \Omega \\ \langle p; i\rangle \bullet \lambda & = & \langle p; i\rangle & \end{array}$$

Furthermore, set $N_{\#} = \{\#_i \mid 1 \le i \le k\}$ *and define a morphism* $\bar\tau$ *from* $(N_{\#} \cup T)$ *to* $(\Sigma - \{\$\})$ *such that* $\bar\tau(a) = a$ *for every* $a \in T$ *and* $\bar\tau(X) = \#$ *for every* $X \in N_{\#}$.

Based on the state G *enters, the following two cases are considered:*

(a) Let $(\#_1, \langle p, 1\rangle)\ {}_k\Rightarrow^m_G (w\alpha\beta, \langle q, \mathscr{O}_{\#}(\bar\tau(\alpha))\rangle \bullet Z)$, *where* $p, q \in Q$, $w \in T^*$, $\alpha \in (N_{\#}(N_{\#} \cup T)^*)^*$, $\beta \in (V - \{\#, \$\})^*$, $Z \in (\Omega \cup \{\lambda\})$, *and* $m \ge 0$. *Then,* $p\#\$ \Rightarrow^*_M qw\bar\tau(\alpha)\$\beta$.

(b) Let $(\#_1, \langle p, 1\rangle)\ {}_k\Rightarrow^m_G (w\alpha\beta, q_{\text{false}})$, *where* $p \in Q$, $w \in T^*$, $\alpha \in (N_{\#}(N_{\#} \cup T)^*)^*$, $\beta \in (V - \{\#, \$\})^*$, *and* $m \ge 0$. *Then,* $p\#\$ \Rightarrow^*_M \bar q w\bar\tau(\alpha)\$\beta$, *where* $\bar q \in Q$, $\beta = z_1 Y z_2 A z_3$, $Y, A \in (V - \Sigma)$, $Y \neq A$, $z_1 \in (\Sigma - \{\#, \$\})^*$, $z_2 \in (V - \{A, \#, \$\})^*$, $z_3 \in (V - \{\#, \$\})^*$, *and there is a rule* $\bar r$: $\bar q\$A \to q'\#\$ \in R$, $q' \in Q$, *such that* $\bar r$ *is not applicable on* $\bar q w\bar\tau(\alpha)\$\beta$.

Proof. This claim is proved by induction on $m \ge 0$.

Basis. Let $m = 0$, so $(\#_1, \langle p; 1\rangle)\ {}_k\Rightarrow^0_G (\#_1, \langle p; 1\rangle)$, where $w = \varepsilon$, $\alpha = \#_1$, $\beta = \varepsilon$, and $Z = \lambda$. Then, $p\#\$ \Rightarrow^*_M p\#\$$ and the basis holds. Observe that the basis also holds for the case (b) of this claim. Since $\langle p; 1\rangle \neq q_{\text{false}}$, $(\#_1, \langle p; 1\rangle)\ {}_k\not\Rightarrow^0_G (\#_1, q_{\text{false}})$ and the implication is true by default.

Induction Hypothesis. Suppose that the claim holds for all $0 \le m \le l$, where l is a non-negative integer.

Induction Step. Let $(\#_1, \langle p; 1\rangle)\ {}_k\Rightarrow^{l+1}_G (w\alpha\beta, \mathscr{X})$, where $p \in Q$, $w \in T^*$, $\alpha \in (N_{\#}(N_{\#} \cup T)^*)^*$, $\beta \in (V - \{\#, \$\})^*$, and either $\mathscr{X} = \langle q; \mathscr{O}_{\#}(\bar\tau(\alpha))\rangle \bullet Z$, $q \in Q$, $Z \in (\Omega \cup \{\lambda\})$, or

$\mathscr{X} = q_{\text{false}}$. Since $l + 1 \geq 1$, express $(\#_1, \langle p; 1\rangle)\ {}_k\!\Rightarrow_G^{l+1}\ (w\alpha\beta, \mathscr{X})$ as $(\#_1, \langle p; 1\rangle)\ {}_k\!\Rightarrow_G^{l}$ $(w'u\mathrm{A}_\# vz, \mathscr{X}')\ {}_k\!\Rightarrow_G (w'uxv'\beta, \mathscr{X})$, where $w' \in \mathrm{T}^*$, $u \in (\mathrm{N}_\#(\mathrm{N}_\# \cup \mathrm{T})^*)^*$, $\mathrm{A}_\# \in (\mathrm{N}_\# \cup \{\varepsilon\})$, $x, v, v' \in (\mathrm{N}_\# \cup \mathrm{T})^*$, $z \in (\mathrm{V} - \{\#, \$\})^*$, $\hat{w}\alpha = uxv'$, $w = w'\hat{w}$, $\hat{w} \in \mathrm{T}^*$, and either $\mathscr{X}' = \langle t; \mathscr{O}_\#(\bar{\tau}(u\mathrm{A}_\# v))\rangle \bullet \mathrm{Z}'$, $t \in \mathrm{Q}$, $\mathrm{Z}' \in (\Omega \cup \{\lambda\})$, or $\mathscr{X}' = q_{\text{false}}$.

By the induction hypothesis,

$$p\#\$ \Rightarrow_M^* \chi$$

where $\chi = tw'\bar{\tau}(u\mathrm{A}_\# v)\$z$, if $\mathscr{X}' = \langle t; \mathscr{O}_\#(\bar{\tau}(u\mathrm{A}_\# v))\rangle \bullet \mathrm{Z}'$, or $\chi = \bar{q}w'\bar{\tau}(u\mathrm{A}_\# v)\$z$, if $\mathscr{X}' = q_{\text{false}}$, and there is a rule $\bar{r}$: $\bar{q}\$\mathrm{A} \rightarrow q'\#\$ \in \mathrm{R}$ such that $\bar{r}$ is not applicable on χ, where $\bar{q}, q' \in \mathrm{Q}$, $z = z_1\mathrm{Y}z_2\mathrm{A}z_3$, $\mathrm{Y}, \mathrm{A} \in (\mathrm{V} - \Sigma)$, $\mathrm{Y} \neq \mathrm{A}$, $z_1 \in (\Sigma - \{\#, \$\})^*$, $z_2 \in (\mathrm{V} - \{\mathrm{A}, \#, \$\})^*$, and $z_3 \in (\mathrm{V} - \{\#, \$\})^*$. Set $\kappa = \mathscr{O}_\#(\bar{\tau}(u\mathrm{A}_\# v))$. Based on the form of the applied rules, G can perform

$$(w'u\mathrm{A}_\# vz, \mathscr{X}')\ {}_k\!\Rightarrow_G (w'uxv'\beta, \mathscr{X})$$

according to one of the following cases:

Case 1. G performs $(w'u\mathrm{A}_\# vz, \mathscr{X}')\ {}_k\!\Rightarrow_G (w'uxv'\beta, \mathscr{X})$ using $(\#_1, \langle t; \kappa\rangle) \rightarrow (\#_1, \langle t; \kappa; r\rangle) \in \mathrm{P}$, where $r \in \mathrm{R}$. In this case $\mathscr{X}' = \langle t; \mathscr{O}_\#(\bar{\tau}(u\mathrm{A}_\# v))\rangle \bullet \mathrm{Z}'$, $\mathscr{X} = \langle t; \mathscr{O}_\#(\bar{\tau}(uxv))\rangle \bullet \mathrm{Z}$, $t = q$, $\mathrm{Z}' = \lambda$, $\mathrm{Z} = r$, $\mathscr{O}_\#(\bar{\tau}(u)) = 0$, $\mathrm{A}_\# = \#_1 = x$, $v = v'$, $\mathscr{O}_\#(\bar{\tau}(v)) = \kappa - 1$, and $z = \beta$. Thus,

$$tw'\bar{\tau}(u\mathrm{A}_\# v)\$z \Rightarrow_M^* qw'\bar{\tau}(uxv)\$z \Rightarrow_M^* qw\bar{\tau}(\alpha)\$\beta$$

Case 2. G performs $(w'u\mathrm{A}_\# vz, \mathscr{X}')\ {}_k\!\Rightarrow_G (w'uxv'\beta, \mathscr{X})$ using $(\#_\kappa, \langle t; \kappa; r\rangle) \rightarrow (x, \langle q; \kappa - 1\rangle) \in \mathrm{P}$, where $r \in \mathrm{R}$ and $x \in \mathrm{T}^*$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_\#(\bar{\tau}(u\mathrm{A}_\# v))\rangle \bullet \mathrm{Z}'$, $\mathscr{X} = \langle q; \mathscr{O}_\#(\bar{\tau}(uxv))\rangle \bullet \mathrm{Z}$, $\mathrm{Z}' = r$, $\mathrm{Z} = \lambda$, $\mathscr{O}_\#(\bar{\tau}(u)) = \kappa - 1$, $\mathrm{A}_\# = \#_\kappa$, $v = v'$, $\mathscr{O}_\#(\bar{\tau}(v)) = 0$, and $z = \beta$. Based on the construction of P, r is of the form $t\,{}_\kappa\# \rightarrow qx$ and therefore

$$tw'\bar{\tau}(u\mathrm{A}_\# v)\$z \Rightarrow_M qw'\bar{\tau}(uxv)\$z \Rightarrow_M^* qw\bar{\tau}(\alpha)\$\beta$$

Case 3. G performs $(w'u\mathrm{A}_\# vz, \mathscr{X}')\ {}_k\!\Rightarrow_G (w'uxv'\beta, \mathscr{X})$ using $(\#_n, \langle t; \kappa; r\rangle) \rightarrow (x, \langle q; \kappa - 1; [\![r, 1]\!]\rangle) \in \mathrm{P}$, where $1 \leq n \leq \kappa - 1$, $r \in \mathrm{R}$, $x \in \mathrm{T}^*$, and r is of the form $t\,{}_n\# \rightarrow qx$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_\#(\bar{\tau}(u\mathrm{A}_\# v))\rangle \bullet \mathrm{Z}'$, $\mathscr{X} = \langle q; \kappa - 1\rangle \bullet \mathrm{Z}$, $\mathrm{Z}' = r$, $\mathrm{Z} = [\![r, 1]\!]$, $\mathscr{O}_\#(\bar{\tau}(u)) = n - 1$, $\mathrm{A}_\# = \#_n$, $v = v'$, $\mathscr{O}_\#(\bar{\tau}(v)) = \kappa - n$, and $z = \beta$. Thus,

$$tw'\bar{\tau}(u\mathrm{A}_\# v)\$z \Rightarrow_M qw'\bar{\tau}(uxv)\$z \Rightarrow_M^* qw\bar{\tau}(\alpha)\$\beta$$

Case 4. G performs $(w'u\mathrm{A}_\# vz, \mathscr{X}')\ {}_k\!\Rightarrow_G (w'uxv'\beta, \mathscr{X})$ using

$$(\#_{n+i}, \langle t; \kappa; [\![r, i]\!]\rangle) \rightarrow (\#_{n+i-1}, \langle t; \kappa; [\![r, i + 1]\!]\rangle) \in \mathrm{P}$$

where $1 \leq n \leq \kappa$, $1 \leq i \leq \kappa - n$, $2 \leq n + i \leq \kappa$, and $r \in \mathrm{R}$ is a rule of the form $t'\,{}_n\# \rightarrow tx'$ with $t' \in \mathrm{Q}$ and $x' \in \mathrm{T}^*$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_\#(\bar{\tau}(u\mathrm{A}_\# v))\rangle \bullet \mathrm{Z}'$, $\mathscr{X} = \langle q; \kappa\rangle \bullet \mathrm{Z}$, $t = q$, $\mathrm{Z}' = [\![r, i]\!]$, $\mathrm{Z} = [\![r, i + 1]\!]$, $\mathscr{O}_\#(\bar{\tau}(u)) = (n + i) - 1$, $\mathrm{A}_\# = \#_{n+i}$, $x = \#_{n+i-1}$, $v = v'$, $\mathscr{O}_\#(\bar{\tau}(v)) = \kappa - (n + i)$, and $z = \beta$. Clearly, $\bar{\tau}(\#_{n+i}) = \bar{\tau}(\#_{n+i-1})$ and then

$$tw'\bar{\tau}(u\mathrm{A}_\# v)\$z \Rightarrow_M^* qw'\bar{\tau}(uxv)\$z \Rightarrow_M^* qw\bar{\tau}(\alpha)\$\beta$$

Case 5. G performs $(w'uA_{\#}vz, \mathscr{X}')\ {}_{k}\Rightarrow_{G} (w'uxv'\beta, \mathscr{X})$ using $(\#_{\kappa+1}, \langle t; \kappa; [\![r, \kappa - n + 1]\!]\rangle) \rightarrow (\#_{\kappa}, \langle t; \kappa\rangle) \in P$, where $1 \leq n \leq \kappa$ and $r \in R$ is of the form $t'\,{}_{n}\# \rightarrow tx'$ with $t' \in Q$ and $x' \in T^*$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_{\#}(\bar{\tau}(uA_{\#}v))\rangle \bullet Z'$, $\mathscr{X} = \langle q; \kappa\rangle \bullet Z$, $t = q$, $Z' = [\![r, \kappa - n + 1]\!]$, $Z = \lambda$, $\mathscr{O}_{\#}(\bar{\tau}(u)) = \kappa - 1$, $A_{\#} = \#_{\kappa+1}$, $x = \#_{\kappa}$, $v = v'$, $\mathscr{O}_{\#}(\bar{\tau}(v)) = 0$, and $z = \beta$. Clearly, as in the previous case,

$$tw'\bar{\tau}(uA_{\#}v)\$z \Rightarrow_{M}^{*} qw'\bar{\tau}(uxv)\$z \Rightarrow_{M}^{*} qw\bar{\tau}(\alpha)\$\beta$$

Case 6. G performs $(w'uA_{\#}vz, \mathscr{X}')\ {}_{k}\Rightarrow_{G} (w'uxv'\beta, \mathscr{X})$ using

$$(\#_{n}, \langle t; \kappa; r\rangle) \rightarrow (x, \langle q; \kappa\rangle) \in P$$

where $1 \leq n \leq \kappa$, $r \in R$, and $x = x_1\#_{n}x_2$ with $x_1, x_2 \in T^*$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_{\#}(\bar{\tau}(uA_{\#}v))\rangle \bullet Z'$, $\mathscr{X} = \langle q; \kappa\rangle \bullet Z$, $Z' = r$, $Z = \lambda$, $\mathscr{O}_{\#}(\bar{\tau}(u)) = n - 1$, $A_{\#} = \#_{n}$, $v = v'$, $\mathscr{O}_{\#}(\bar{\tau}(v)) = \kappa - n$, and $z = \beta$. Following the construction of P, r is of the form $t\,{}_{n}\# \rightarrow q\bar{\tau}(x)$ and hence

$$tw'\bar{\tau}(uA_{\#}v)\$z \Rightarrow_{M} qw'\bar{\tau}(uxv)\$z \Rightarrow_{M}^{*} qw\bar{\tau}(\alpha)\$\beta$$

Case 7. G performs $(w'uA_{\#}vz, \mathscr{X}')\ {}_{k}\Rightarrow_{G} (w'uxv'\beta, \mathscr{X})$ using

$$(\#_{n}, \langle t; \kappa; r\rangle) \rightarrow (\#_{n}, \langle t; \kappa; [\![r, 1]\!]\rangle) \in P$$

where $1 \leq n \leq \kappa$ and $r \in R$ is of the form $t\,{}_{n}\# \rightarrow q'x'$ with $q' \in Q$, $x' \in (\Sigma - \{\$\})^*$, and $\mathscr{O}_{\#}(x') \geq 2$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_{\#}(\bar{\tau}(uA_{\#}v))\rangle \bullet Z'$, $\mathscr{X} = \langle q; \kappa\rangle \bullet Z$, $t = q$, $Z' = r$, $Z = [\![r, 1]\!]$, $\mathscr{O}_{\#}(\bar{\tau}(u)) = n - 1$, $A_{\#} = x = \#_{n}$, $v = v'$, $\mathscr{O}_{\#}(\bar{\tau}(v)) = \kappa - n$, and $z = \beta$. Clearly,

$$tw'\bar{\tau}(uA_{\#}v)\$z \Rightarrow_{M}^{*} qw'\bar{\tau}(uxv)\$z \Rightarrow_{M}^{*} qw\bar{\tau}(\alpha)\$\beta$$

Case 8. G performs $(w'uA_{\#}vz, \mathscr{X}')\ {}_{k}\Rightarrow_{G} (w'uxv'\beta, \mathscr{X})$ using $(\#_{\kappa-i}, \langle t; \kappa; [\![r, i + 1]\!]\rangle) \rightarrow (\#_{\kappa+\eta-i}, \langle t; \kappa; [\![r, i + 2]\!]\rangle) \in P$, where $0 \leq i \leq \kappa - n - 1$, $1 \leq n \leq \kappa - 1$, $r \in R$ is of the form $t\,{}_{n}\# \rightarrow q'x'$ with $q' \in Q$, $x' \in (\Sigma - \{\$\})^*$, $\mathscr{O}_{\#}(x') \geq 2$, and $\eta = \mathscr{O}_{\#}(x') - 1$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_{\#}(\bar{\tau}(uA_{\#}v))\rangle \bullet Z'$, $\mathscr{X} = \langle q; \kappa\rangle \bullet Z$, $t = q$, $Z' = [\![r, i + 1]\!]$, $Z = [\![r, i + 2]\!]$, $\mathscr{O}_{\#}(\bar{\tau}(u)) = (\kappa - i) - 1$, $A_{\#} = \#_{\kappa-i}$, $x = \#_{\kappa+\eta-i}$, $v = v'$, $\mathscr{O}_{\#}(\bar{\tau}(v)) = i$, and $z = \beta$. As $\bar{\tau}(A_{\#}) = \bar{\tau}(x)$, the following holds

$$tw'\bar{\tau}(uA_{\#}v)\$z \Rightarrow_{M}^{*} qw'\bar{\tau}(uxv)\$z \Rightarrow_{M}^{*} qw\bar{\tau}(\alpha)\$\beta$$

Case 9. G performs $(w'uA_{\#}vz, \mathscr{X}')\ {}_{k}\Rightarrow_{G} (w'uxv'\beta, \mathscr{X})$ using

$$(\#_{n}, \langle t; \kappa; [\![r, \kappa - n + 1]\!]\rangle) \rightarrow (x, \langle q; \kappa + \eta\rangle) \in P$$

where $1 \leq n \leq \kappa$, $x \in (N_{\#} \cup T)^*$, $\eta = \mathscr{O}_{\#}(\bar{\tau}(x)) - 1$, $\eta \geq 1$, and $r \in R$ is of the form $t\,{}_{n}\# \rightarrow q\bar{\tau}(x)$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_{\#}(\bar{\tau}(uA_{\#}v))\rangle \bullet Z'$, $\mathscr{X} = \langle q; \kappa + \eta\rangle \bullet Z$, $Z' = [\![r, \kappa - n + 1]\!]$, $Z = \lambda$, $\mathscr{O}_{\#}(\bar{\tau}(u)) = n - 1$, $A_{\#} = \#_{n}$, $v = v'$, $\mathscr{O}_{\#}(\bar{\tau}(v)) = \kappa - n$, and $z = \beta$. Therefore, it is clear that

$$tw'\bar{\tau}(uA_{\#}v)\$z \Rightarrow_{M} qw'\bar{\tau}(uxv)\$z \Rightarrow_{M}^{*} qw\bar{\tau}(\alpha)\$\beta$$

Case 10. G performs $(w'u\mathrm{A}_{\#}vz, \mathscr{X}')\ {}_k\!\Rightarrow_{\mathrm{G}} (w'uxv'\beta, \mathscr{X})$ using $(\#_\kappa, \langle t; \kappa; r\rangle) \to (y, \langle q; \kappa - 1\rangle) \in \mathrm{P}$, where $r \in \mathrm{R}$ and $y \in (\mathrm{V}' - \mathrm{N}_{\#})^*$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_{\#}(\bar{\tau}(u\mathrm{A}_{\#}v))\rangle \bullet \mathrm{Z}'$, $\mathscr{X} = \langle q; \kappa - 1\rangle \bullet \mathrm{Z}$, $\mathrm{Z}' = r$, $\mathrm{Z} = \lambda$, $\mathscr{O}_{\#}(\bar{\tau}(u)) = \kappa - 1$, $\mathrm{A}_{\#} = \#_\kappa$, $x = v = v' = \varepsilon$, and $\beta = yz$. Following the construction of P, r is of the form $t\#\$ \to q\$y$ and

$$tw'\bar{\tau}(u\mathrm{A}_{\#}v)\$z \Rightarrow_{\mathrm{M}} qw'\bar{\tau}(uxv')\$yz \Rightarrow^*_{\mathrm{M}} qw\bar{\tau}(\alpha)\$\beta$$

Case 11. G performs $(w'u\mathrm{A}_{\#}vz, \mathscr{X}')\ {}_k\!\Rightarrow_{\mathrm{G}} (w'uxv'\beta, \mathscr{X})$ using

$$(\mathrm{A}, \langle t; \kappa\rangle) \to (\mathrm{A}, \langle t; \kappa; r\rangle) \in \mathrm{P}$$

where $\mathrm{A} \in (\mathrm{V}' - \mathrm{N}_{\#} - \mathrm{T})$, $\kappa = 0$, and $r \in \mathrm{R}$ is of the form $t\$\mathrm{A} \to q'\#\$$ with $q' \in \mathrm{Q}$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_{\#}(\bar{\tau}(u\mathrm{A}_{\#}v))\rangle \bullet \mathrm{Z}'$, $\mathscr{X} = \langle q; \kappa\rangle \bullet \mathrm{Z}$, $t = q$, $\mathrm{Z}' = \lambda$, $\mathrm{Z} = r$, $\mathrm{A}_{\#} = x = \varepsilon$, $v = v'$, $z = \beta = z_1\mathrm{A}z_2$, $z_1, z_2 \in (\mathrm{V}' - \mathrm{N}_{\#})^*$, and $\mathscr{O}_{\mathrm{V}'-\mathrm{N}_{\#}-\mathrm{T}}(z_1) \le k - 1$. Hence,

$$tw'\bar{\tau}(u\mathrm{A}_{\#}v)\$z \Rightarrow^*_{\mathrm{M}} qw'\bar{\tau}(uxv')\$\beta \Rightarrow^*_{\mathrm{M}} qw\bar{\tau}(\alpha)\$\beta$$

Case 12. G performs $(w'u\mathrm{A}_{\#}vz, \mathscr{X}')\ {}_k\!\Rightarrow_{\mathrm{G}} (w'uxv'\beta, \mathscr{X})$ using

$$(\mathrm{A}, \langle t; \kappa; r\rangle) \to (\mathrm{A}, \langle t; \kappa; [\![r, ¿]\!]\rangle) \in \mathrm{P}$$

where $\mathrm{A} \in (\mathrm{V}' - \mathrm{N}_{\#} - \mathrm{T})$ and $r \in \mathrm{R}$ is of the form $t\$\mathrm{A} \to q'\#\$$ with $q' \in \mathrm{Q}$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_{\#}(\bar{\tau}(u\mathrm{A}_{\#}v))\rangle \bullet \mathrm{Z}'$, $\mathscr{X} = \langle q; \kappa\rangle \bullet \mathrm{Z}$, $t = q$, $\mathrm{Z}' = r$, $\mathrm{Z} = [\![r, ¿]\!]$, $\mathrm{A}_{\#} = x = \varepsilon$, $v = v'$, $z = \beta = z_1\mathrm{A}z_2$, $z_1, z_2 \in (\mathrm{V}' - \mathrm{N}_{\#})^*$, and $\kappa + \mathscr{O}_{\mathrm{V}'-\mathrm{N}_{\#}-\mathrm{T}}(z_1) \le k - 1$. Hence,

$$tw'\bar{\tau}(u\mathrm{A}_{\#}v)\$z \Rightarrow^*_{\mathrm{M}} qw'\bar{\tau}(uxv')\$\beta \Rightarrow^*_{\mathrm{M}} qw\bar{\tau}(\alpha)\$\beta$$

Case 13. G performs $(w'u\mathrm{A}_{\#}vz, \mathscr{X}')\ {}_k\!\Rightarrow_{\mathrm{G}} (w'uxv'\beta, \mathscr{X})$ using

$$(\mathrm{X}, \langle t; \kappa; [\![r, ¿]\!]\rangle) \to (\mathrm{X}, \langle t; \kappa; [\![r, \mathrm{X}]\!]\rangle) \in \mathrm{P}$$

where $\mathrm{X} \in (\mathrm{V}' - \mathrm{N}_{\#} - \mathrm{T})$ and $r \in \mathrm{R}$ is of the form $t\$\mathrm{A}' \to q'\#\$$ with $\mathrm{A}' \in (\mathrm{V}' - \mathrm{N}_{\#} - \mathrm{T})$ and $q' \in \mathrm{Q}$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_{\#}(\bar{\tau}(u\mathrm{A}_{\#}v))\rangle \bullet \mathrm{Z}'$, $\mathscr{X} = \langle q; \kappa\rangle \bullet \mathrm{Z}$, $t = q$, $\mathrm{Z}' = [\![r, ¿]\!]$, $\mathrm{Z} = [\![r, \mathrm{X}]\!]$, $\mathrm{A}_{\#} = x = \varepsilon$, $v = v'$, $z = \beta = z_1\mathrm{X}z_2$, $z_1 \in \mathrm{T}^*$, $z_2 \in (\mathrm{V}' - \mathrm{N}_{\#})^*$, and $\kappa \le k - 1$. Hence,

$$tw'\bar{\tau}(u\mathrm{A}_{\#}v)\$z \Rightarrow^*_{\mathrm{M}} qw'\bar{\tau}(uxv')\$\beta \Rightarrow^*_{\mathrm{M}} qw\bar{\tau}(\alpha)\$\beta$$

Case 14. G performs $(w'u\mathrm{A}_{\#}vz, \mathscr{X}')\ {}_k\!\Rightarrow_{\mathrm{G}} (w'uxv'\beta, \mathscr{X})$ using

$$(\mathrm{Y}, \langle t; \kappa; [\![r, \mathrm{Y}]\!]\rangle) \to (\mathrm{Y}, q_{\mathrm{false}}) \in \mathrm{P}$$

where $\mathrm{Y} \in (\mathrm{V}' - \mathrm{N}_{\#} - \mathrm{T})$ and $r \in \mathrm{R}$ is of the form $t\$\mathrm{A}' \to q'\#\$$ with $\mathrm{A}' \in (\mathrm{V}' - \mathrm{N}_{\#} - \mathrm{T})$, $\mathrm{A}' \neq \mathrm{Y}$, and $q' \in \mathrm{Q}$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_{\#}(\bar{\tau}(u\mathrm{A}_{\#}v))\rangle \bullet \mathrm{Z}'$, $\mathscr{X} = q_{\mathrm{false}}$, $\mathrm{Z}' = [\![r, \mathrm{Y}]\!]$, $\mathrm{A}_{\#} = x = \varepsilon$, $v = v'$, $z = \beta = z_1\mathrm{Y}z_2\mathrm{A}'z_3$, $z_1 \in \mathrm{T}^*$, $z_2 \in (\mathrm{V}' - \mathrm{N}_{\#} - \{\mathrm{A}'\})^*$, $z_3 \in (\mathrm{V}' - \mathrm{N}_{\#})^*$, and $\kappa + \mathscr{O}_{\mathrm{V}'-\mathrm{N}_{\#}-\mathrm{T}}(z_1\mathrm{Y}z_2) \le k - 1$. With $\bar{q} = t$, it follows that

$$tw'\bar{\tau}(u\mathrm{A}_{\#}v)\$z \Rightarrow^*_{\mathrm{M}} tw'\bar{\tau}(uxv')\$\beta \Rightarrow^*_{\mathrm{M}} \bar{q}w\bar{\tau}(\alpha)\$\beta$$

and the rule r: $\bar{q}\$A' \to q'\#\$ \in R$ is not applicable on $\bar{q}w\bar{\tau}(\alpha)\$\beta$.

Case 15. G performs $(w'uA_{\#}vz, \mathscr{X}') \;{}_k\!\Rightarrow_G (w'uxv'\beta, \mathscr{X})$ using

$$(A, \langle t; \kappa; [\![r, A]\!]\rangle) \to (\#_{\kappa+1}, \langle q; \kappa + 1\rangle) \in P$$

where $A \in (V' - N_{\#} - T)$ and $r \in R$ is of the form $t\$A \to q\#\$$. In this case, $\mathscr{X}' = \langle t; \mathscr{O}_{\#}(\bar{\tau}(uA_{\#}v))\rangle \bullet Z'$, $\mathscr{X} = \langle q; \kappa + 1\rangle \bullet Z$, $Z' = [\![r, A]\!]$, $Z = \lambda$, $A_{\#} = x = \varepsilon$, $v' = v\#_{\kappa+1}$, and $z = A\beta$. Therefore,

$$tw'\bar{\tau}(uA_{\#}v)\$z \Rightarrow_M qw'\bar{\tau}(uxv')\$\beta \Rightarrow_M^* qw\bar{\tau}(\alpha)\$\beta$$

□

If $p = s$ and $\alpha = \beta = \varepsilon$ are used in Claim 7.2.5, then $s\#\$ \Rightarrow_M^* qw\$$ implies $(\#_1, \langle s; 1\rangle) \;{}_k\!\Rightarrow_G^* (w, \langle q; 0\rangle)$ which proves $L(M) \subseteq L(G, k)$. Conversely, for $p = s$, $\alpha = \beta = \varepsilon$, and $Z = \lambda$ in Claim 7.2.6, $(\#_1, \langle s; 1\rangle) \;{}_k\!\Rightarrow_G^* (w, \langle q; 0\rangle)$ implies $s\#\$ \Rightarrow_M^* qw\$$ which proves $L(G, k) \subseteq L(M)$. Hence, $L(M) = L(G, k)$ and the lemma holds. □

Theorem 7.2.7. *Let $k \geq 1$. Then, $\mathscr{L}_k(\#\$RS) = \mathbf{ST}_k$.*

Proof. It directly follows from Lemma 7.2.1 and Lemma 7.2.4. □

Next, it will be shown that $\mathscr{L}_k(\#RS)$ is properly included in $\mathscr{L}_k(\#\$RS)$ for every $k \geq 1$.

Theorem 7.2.8. *Let $k \geq 1$. Then, $\mathscr{L}_k(\#RS) \subset \mathscr{L}_k(\#\$RS)$.*

Proof. The inclusion $\mathscr{L}_k(\#RS) \subseteq \mathscr{L}_k(\#\$RS)$ follows directly from the definitions of #-rewriting system of index k and $k\#\$$-rewriting system. Now, a language contained in $\mathscr{L}_k(\#\$RS)$ but not in $\mathscr{L}_k(\#RS)$ needs to be found.

For $k = 1$, such a language is $\mathscr{D}_2$. As $\mathscr{L}_1(\#\$RS) = \mathbf{CF}$ (by [36] and Theorem 7.2.7), $\mathscr{D}_2 \in \mathscr{L}_1(\#\$RS)$, but $\mathscr{D}_2 \notin \mathscr{L}_1(\#RS)$ (see page 169 in [21]).

For $k \geq 2$, let $\Sigma_k = \{a_i \mid 1 \leq i \leq 4k - 2\}$ be an alphabet. Define a language L_k over Σ_k as

$$L_k = \{a_1^i a_2^i \dots a_{4k-2}^i \mid i \geq 1\}$$

By Theorem 4 in [36], $L_k \in \mathbf{ST}_k$ and since $\mathscr{L}_k(\#\$RS) = \mathbf{ST}_k$, $L_k \in \mathscr{L}_k(\#\$RS)$ as well.

Since ${}_k\mathbf{PRG} = \mathscr{L}_k(\#RS)$, as recalled in Theorem 3.1.10, it will be be demonstrated by Lemma 3.1.4 that $L_k \notin \mathscr{L}_k(\#RS)$. Assume that $L_k \in \mathscr{L}_k(\#RS)$. Therefore, there exists $z \in L_k$ such that

$$z = u_1v_1w_1x_1u_2v_2w_2x_2 \dots u_lv_lw_lx_lu_{l+1}$$

with $l \leq k$, $|v_1x_1v_2x_2 \dots v_lx_l| > 0$, and

$$u_1v_1^iw_1x_1^iu_2v_2^iw_2x_2^i \dots u_lv_l^iw_lx_l^iu_{l+1} \in L_k$$

$$\begin{array}{ccccccc}
\mathscr{L}_1(\#\$\mathrm{RS}) & \subset & \mathscr{L}_2(\#\$\mathrm{RS}) & \subset & \cdots & \subset & \mathscr{L}_n(\#\$\mathrm{RS}) \\
\cup & & \cup & & & & \cup \\
\mathscr{L}_1(\#\mathrm{RS}) & \subset & \mathscr{L}_2(\#\mathrm{RS}) & \subset & \cdots & \subset & \mathscr{L}_n(\#\mathrm{RS})
\end{array}$$

Figure 3. The relations between #-rewriting systems with finite index and k#$-rewriting systems.

for every $i \geq 1$. Now, consider the following cases:

- There exists $y \in \{v_1, x_1, v_2, x_2, \ldots, v_l, x_l\}$ such that $\mathrm{card}(\mathrm{alph}(y)) \geq 2$. In this case, there exists $i \geq 1$ such that

$$u_1 v_1^i w_1 x_1^i u_2 v_2^i w_2 x_2^i \ldots u_l v_l^i w_l x_l^i u_{l+1} \notin \mathrm{L}_k$$

- All $v_1, x_1, v_2, x_2, \ldots, v_l, x_l$ are words over unary alphabet. As for $k \geq 2$ it is always true that $4k - 2 > 2k$, there will always be symbols from $\mathrm{alph}(z)$ that are not contained in $\mathrm{alph}(v_1 x_1 v_2 x_2 \ldots v_l x_l)$. Hence there must exist $i \geq 1$ such that

$$u_1 v_1^i w_1 x_1^i u_2 v_2^i w_2 x_2^i \ldots u_l v_l^i w_l x_l^i u_{l+1} \notin \mathrm{L}_k$$

Such $z \in \mathrm{L}_k$ does not exist and therefore $\mathrm{L}_k \notin \mathscr{L}_k(\#\mathrm{RS})$ for every $k \geq 2$. $\square$

The relationship between infinite hierarchies of #-rewriting systems of finite index and k#$-rewriting systems is summed up in Figure 3.

Part III
Conclusion

This part, consisting only of Chapter 8, closes this thesis by summing up achieved theoretical results and encountered open problems. It also gives suggestions for further research and application areas.

Chapter 8
Conclusion

In this thesis four new language models were presented—state-synchronized automata systems, unlimited deep pushdown automata, jumping pure grammars, and k#\$-rewriting systems.

As summarized in Theorem 4.2.16, state-synchronized automata systems with two or more pushdown components are capable of accepting every recursively enumerable language, both in a deterministic and nondeterministic way. Considering the nondeterministic way, this also holds for the case when components are one-turn pushdown automata. However, the accepting power of state-synchronized automata systems with two or more one-turn pushdown automata components that work in a deterministic way remains as an open problem.

Recursively enumerable languages can also be accepted by both types of unlimited deep pushdown automata, as demonstrated by Theorem 5.1.9 and Theorem 5.2.3. Both types can be further restricted to accept only context-sensitive languages by forbidding the use of erasing rules (see Theorem 5.1.10 and Theorem 5.2.4).

In Figure 1 and Table 1, mutual relations of language families generated by jumping pure grammars with context-free rules are visualized. Clearly, jumping rewriting has no impact on generative capacity when only unary alphabets are considered (see Figure 2). Some mutual relations between language families depicted in Figure 1 are not yet known. Below can be found the list of all open problems concerning jumping pure grammars:

- Is it true that $(\mathbf{PPCF} \cap \mathbf{JSPCF}) - (\mathbf{CF} \cup \mathbf{JPPCF}) \neq \emptyset$ (Open Problem 6.3.2)?
- Is it true that $(\mathbf{PPCF} \cap \mathbf{JSPCF} \cap \mathbf{JPPCF}) - \mathbf{CF} \neq \emptyset$ (Open Problem 6.3.3)?
- Is it true that $(\mathbf{SPCF} \cap \mathbf{JPPCF}) - \mathbf{JSPCF} \neq \emptyset$ (Open Problem 6.3.8)?
- Is it true that $(\mathbf{PPCF} \cap \mathbf{CF} \cap \mathbf{JSPCF}) - (\mathbf{SPCF} \cup \mathbf{JPPCF}) \neq \emptyset$ (Open Problem 6.3.10)?
- Is it true that $(\mathbf{PPCF} \cap \mathbf{CF} \cap \mathbf{JSPCF} \cap \mathbf{JPPCF}) - \mathbf{SPCF} \neq \emptyset$ (Open Problem 6.3.11)?
- Is it true that $(\mathbf{CF} \cap \mathbf{JSPCF}) - (\mathbf{PPCF} \cup \mathbf{JPPCF}) \neq \emptyset$ (Open Problem 6.3.14)?
- Is it true that $\mathbf{JSPCF} - (\mathbf{CF} \cup \mathbf{PPCF} \cup \mathbf{JPPCF}) \neq \emptyset$ (Open Problem 6.3.17)?
- Let $\mathrm{X} \in \{\mathbf{SP}, \mathbf{JSP}, \mathbf{PP}, \mathbf{JPP}\}$. Is the inclusion $\mathrm{X}^{-\varepsilon} \subseteq \mathrm{X}$, in fact, proper (Open Problem 6.4.2)?

- What is the relation between $\mathbf{JSPCF}^{-\varepsilon}$ and $\mathbf{JPPCF}^{-\varepsilon}$ (Open Problem 6.4.5)?
- What is the relation between $\mathbf{JSPCF}^{-\varepsilon}$ and **JPPCF** (Open Problem 6.4.6)?

k#$-rewriting systems, as proven by Theorem 7.2.7, have the same generative capacity as state grammars working in k-limited way. By Theorem 7.2.8, it was proved that #-rewriting systems of index k are properly included in k#$-rewriting systems, which implies ${}_k\mathbf{PRG} \subset \mathbf{ST}_k$.

8.1 Suggestions for Further Research

Besides the open problems listed above, several ideas concerning further investigation of presented language models have arisen. The present section gives a survey of possible directions of further research.

State-Synchronized Automata Systems

Chapter 4 gives a definition of state-synchronized automata systems and studies their accepting power. However, it does not discuss the possible application areas of such systems. As communication between components is performed in a simple way, it can be easily implemented on hardware level. Therefore, one of the possible directions of further research can be designing a method of conversion of a concurrent system, which is described in a suitable modeling language, to state-synchronized automata system and then converting this state-synchronized automata system to a digital circuit. Focus can be put on the economical way of these conversions, like the number of states and transitions of particular components, the size of a control language, or the complexity of the final digital circuit.

Another area of application of state-synchronized automata systems worth studying can be compiler design. For instance:

- Augmenting LL grammars with respect to state-synchronized automata systems.
- Studying power of state-synchronized automata systems if their first pushdown component can accept only LL languages.
- Using state-synchronized automata systems the models for parallel syntax and semantic analysis.

Finally, state-synchronized automata systems can be studied in association with models for concurrency, e.g. transition systems (see [96]).

Unlimited Deep Pushdown Automata

Recall the definition of direct expansion move of absolutely unlimited deep pushdown automata from Definition 5.1.1:

> *The* M*-based relation of direct expansion move,* ${}_{e}^{a}\vdash_{\mathrm{M}}$*, over* Ξ *is defined as follows:*
> $$(p, w, u\mathrm{A}v) \; {}_{e}^{a}\vdash_{\mathrm{M}} \; (q, w, uxv)$$
> *if and only if* $p\mathrm{A} \rightarrow qx \in \mathrm{R}$, $\mathrm{A} \notin \mathrm{alph}(u)$ *and for every* $\mathrm{A}' \in (\mathrm{alph}(u) - \Sigma)$*,* $p\mathrm{A}' \rightarrow q'x' \notin \mathrm{R}$*, where* $p, q, q' \in \mathrm{Q}$*,* $w \in \Sigma^*$*,* $\mathrm{A} \in (\Gamma - \Sigma)$*, and* $u, v, x, x' \in \Gamma^*$*.*

Now, introduce three modifications of direct expansion move defined above:

(a) The M-based relation of direct expansion move, ${}_{e}^{a}\vdash_{\mathrm{M}}$, over Ξ is defined as follows:

$$(p, w, u\mathrm{A}v) \; {}_{e}^{a}\vdash_{\mathrm{M}} \; (q, w, uxv)$$

if and only if $p\mathrm{A} \rightarrow qx \in \mathrm{R}$ and $\mathrm{A} \notin \mathrm{alph}(u)$, where $p, q \in \mathrm{Q}$, $w \in \Sigma^*$, $\mathrm{A} \in (\Gamma - \Sigma)$, and $u, v, x \in \Gamma^*$.

(b) The M-based relation of direct expansion move, ${}_{e}^{a}\vdash_{\mathrm{M}}$, over Ξ is defined as follows:

$$(p, w, u\mathrm{A}v) \; {}_{e}^{a}\vdash_{\mathrm{M}} \; (q, w, uxv)$$

if and only if $p\mathrm{A} \rightarrow qx \in \mathrm{R}$, where $p, q \in \mathrm{Q}$, $u, w \in \Sigma^*$, $\mathrm{A} \in (\Gamma - \Sigma)$, and $v, x \in \Gamma^*$.

(c) The M-based relation of direct expansion move, ${}_{e}^{a}\vdash_{\mathrm{M}}$, over Ξ is defined as follows:

$$(p, w, u\mathrm{A}v) \; {}_{e}^{a}\vdash_{\mathrm{M}} \; (q, w, uxv)$$

if and only if $p\mathrm{A} \rightarrow qx \in \mathrm{R}$, where $p, q \in \mathrm{Q}$, $w \in \Sigma^*$, $\mathrm{A} \in (\Gamma - \Sigma)$, and $u, v, x \in \Gamma^*$.

How will the accepting power of absolutely unlimited deep pushdown automata change in the case of (a), (b), and (c)?

Other areas of interest concerning unlimited deep pushdown automata could focus on studying their normal forms or how the number of turns on a pushdown will change their accepting power.

Jumping Pure Grammars

Regarding jumping pure grammars, plenty of ideas to advance the topic come to mind. Other than resolving stated open problems and studying jumping pure grammars with other than context-free rules, there are also areas where further investigation of jumping pure grammars can be directed, namely:

(I) *Closure Properties.* Given the families of languages from Definition 6.1.2, which ones of them are closed under intersection, union, concatenation, complement, and other operations over languages?

(II) *New Relations of Direct Derivation.* Let $\mathrm{G} = (\Sigma, \mathrm{P}, \sigma)$ be a pure grammar. Define two G-based relations of direct derivation over Σ^*, ${}_{jf}\Rightarrow_{\mathrm{G}}$ (*jumping folding*) and ${}_{sw}\Rightarrow_{\mathrm{G}}$ (*swapping*), as follows:

- $u_1xu_2 \; {}_{jf}\Rightarrow_{\mathrm{G}} \; v_1yv_2yv_3$ if and only if $u_1u_2 = v_1v_2v_3$, $x \to y \in \mathrm{P}$, and $u_1, u_2, v_1, v_2, v_3, x, y \in \Sigma^*$;
- $uxtyv \; {}_{sw}\Rightarrow_{\mathrm{G}} \; uytxv$ if and only if $x \to y \in \mathrm{P}$ and $u, v, t, x, y \in \Sigma^*$.

What language families are generated by jumping pure grammars working under direct derivation relations defined above?

(III) *Regulated Rewriting.* Consider using regulating devices discussed in Chapter 3, like prescribed sequences of rules or context conditions, for jumping pure grammars. How does it change their power?

Furthermore, consider using *Levenshtein distance* (see [13]) as a regulating device. Let $\mathrm{G} = (\Sigma, \mathrm{P}, \sigma)$ be a pure grammar. For $u, v \in \Sigma^*$, let $\mathrm{d}_\ell(u, v)$ denote the Levenshtein distance between u and v (i.e. the minimum number of insertions, deletions, or substitutions of symbols needed to transform u to v). Furthermore, let $h \in \{lj, rj, j, jp\}$ and let $k \in \mathbb{N}$. What is the generative capacity of pure grammars generating their words under ${}_h\Rightarrow$ such that $u \; {}_h\Rightarrow \; v$ implies $\mathrm{d}_\ell(u, v) \leq k$?

(IV) *Applications Perspectives.* In Chapter 6, jumping pure grammars were studied only theoretically. It is then right to ask how they can be used practically. One area where jumping pure grammars can find their place is genetic algorithms.

Genetic algorithms (see [45]) were inspired by natural evolution and are usually used to find an approximate solution of NP-hard problems. In a genetic algorithm, a solution to a problem is represented by a *chromosome*. A chromosome is a sequence of *genes*, where gene can represent a property or a parameter, depending on what kind of problem is being solved. A set of chromosomes is called a *population*. The important part of a genetic algorithm is a *fitness function*. Its purpose is to map a chromosome to its *fitness value* and its definition depends on the given problem. Based on fitness value, it is decided whether an individual's chromosome survives or not.

The genetic algorithm works as follows. At start, initial population is generated. Then individuals with the best fitness value are selected. These individuals will become *parents* of a new population. Two parental chromosomes, uv and xy, are combined to make a new pair of chromosomes, xv and uy, called *offspring*. This process is called *crossover*. Offspring are then included in the new population. Sometimes, some offspring genes are randomly changed. This process is called *mutation* and it introduces a diversity element. The new population is then used in the next iteration of the algorithm. The algorithm terminates when there are no notable differences between the new and old population.

From the formal language theory point of view, genes can be represented by symbols, chromosomes by words, and population by finite languages. Suppose that individuals' chromosomes consist of genes a, b, and c and only individuals with chromosomes with the number of *a*s equal to the number of *b*s can survive. This can be expressed by the language

$$\mathrm{L}_f = \{w \mid \mathscr{O}_a(w) = \mathscr{O}_b(w), w \in \{a, b, c\}^+\}$$

generated by the jumping sequential context-free pure grammar

$$\mathrm{G}_f = (\{a, b, c\}, \{c \to cc, c \to ab, a \to a, b \to b\}, c)$$

Of course, L_f can be also generated by the unrestricted grammar

$$\mathrm{H}_f = (\{\mathrm{S}, \mathrm{A}, \mathrm{B}, a, b, c\}, \{a, b, c\}, \mathrm{P}, \mathrm{S})$$

with P containing rules

$$\begin{array}{lll} \mathrm{S} \to \mathrm{SS} & \mathrm{S} \to \mathrm{AB} & \mathrm{S} \to c \\ \mathrm{SA} \to \mathrm{AS} & \mathrm{SB} \to \mathrm{BS} & \mathrm{AS} \to \mathrm{SA} \\ \mathrm{AB} \to \mathrm{BA} & \mathrm{BS} \to \mathrm{SB} & \mathrm{BA} \to \mathrm{AB} \\ \mathrm{A} \to a & \mathrm{B} \to b & \end{array}$$

However, as can be seen, G_f contains only 4 rules and hence describes L_f in a more economical way than H_f.

(V) *Automata Counterparts.* Related to (IV), what kind of automata are suitable to effectively recognize languages generated by jumping pure grammars?

k#$-Rewriting Systems

The proposed areas considered for further investigation of *k*#$-rewriting systems are:

(I) Since a new characterization of $\mathbf{ST}_k$ has been given, for some $k \geq 1$, what is the relationship between *k*#$-rewriting systems and generalized #-rewriting systems (studied in Sections 4.1.4 and 5.1.3 of [47])?

(II) Considering [93], *k*#$-rewriting systems can be further discussed in terms of picture languages or 2D languages.

(III) Related to the study given in [94], what is the relationship between multi-head finite automata and *k*#$-rewriting systems?

Bibliography

1. S. Abraham. Some questions of phrase-structure grammars. *Computational Linguistics*, 4:61–70, 1965.
2. A. V. Aho. Indexed grammars. An extension of context-free grammars. *Journal of the ACM*, 15:647–671, 1968.
3. A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Boston, 2nd edition, 2007.
4. M. H. ter Beek, E. Csuhaj-Varjú, and V. Mitrana. Teams of pushdown automata. *International Journal of Computer Mathematics*, 81(2):141–156, 2004.
5. F. J. Brandenburg. On context-free grammars with regular control set. In *Proceedings of the 7th Conference on Automata and Formal Languages*, 1993.
6. M. Čermák. *Formal Systems Based upon Automata and Grammars.* PhD thesis, Faculty of Information Technology, Brno University of Technology, Brno, 2012.
7. M. Čermák and A. Meduna. n-accepting restricted pushdown automata systems. In *13th International Conference on Automata and Formal Languages*, pages 168–183. Computer and Automation Research Institute, Hungarian Academy of Sciences, Nyíregyháza, Hungary, 2011.
8. H. Chigahara, S. Z. Fazekas, and A. Yamamura. One-way jumping finite automata. *International Journal of Foundations of Computer Science*, 27:391–405, 2016.
9. N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.
10. N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.
11. A. Church. An unsolvable problem of elementary number theory. *The American Journal of Mathematics*, 58(2):345–363, 1936.
12. A. B. Cremers and O. Mayer. On vector languages. *Journal of Computer and System Sciences*, 8(8):158–166, 1974.
13. M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings.* Cambridge University Press, New York, 2007.
14. E. Csuhaj-Varjú and J. Dassow. On cooperating/distributed grammar systems. *Journal of Information Processing and Cybernetics*, 26(1–2):49–63, 1990.

15. E. Csuhaj-Varjú, J. Dassow, J. Kelemen, and G. Păun. *Grammar Systems: A Grammatical Approach to Distribution and Cooperation.* Gordon and Breach Science Publishers, Singapore, 1994.
16. E. Csuhaj-Varjú, J. Kelemen, and G. Păun. Grammar systems with WAVE-like communication. *Computers and Artificial Intelligence*, 15(5):419–436, 1996.
17. E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana, and G. Vaszil. Parallel communicating pushdown automata systems. *International Journal of Foundations of Computer Science*, 11(4):631–650, 2000.
18. E. Csuhaj-Varjú, V. Mitrana, and G. Vaszil. Distributed pushdown automata systems: computational power. In Z. Ésik and Z. Fülöp, editors, *DLT '03: Proceedings of the 7th International Conference on Developments in Language Theory*, volume 2710 of *LNCS*, pages 218–229. Springer, Berlin, Heidelberg, 2003.
19. K. Čulík and H. A. Maurer. Tree controlled grammars. *Computing*, 19:129–139, 1977.
20. J. Dassow. Grammars with regulated rewriting. In C. Martín-Vide, V. Mitrana, and G. Păun, editors, *Formal Languages and Applications*, volume 148 of *Studies in Fuzziness and Soft Computing*, pages 249–274. Springer-Verlag, Berlin, Heidelberg, 2004.
21. J. Dassow and G. Păun. *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin, 1989.
22. H. Fernau, M. Paramasivan, and M. L. Schmid. Jumping finite automata: Characterizations and complexity. In F. Drewes, editor, *Proceedings of 20th International Conference on Implementation and Application of Automata (CIAA 2015)*, pages 89–101, 2015.
23. H. Fernau, M. Paramasivan, M. L. Schmid, and V. Vorel. Characterization and complexity results on jumping finite automata. *Theoretical Computer Science*, 679:31–52, 2017.
24. M. J. Fisher. Grammars with macro-like productions. In *SWAT '68: Proceedings of the 9th Annual Symposium on Switching and Automata Theory*, pages 131–142. IEEE Computer Society, 1968.
25. I. Friš. Grammars with partial ordering of the rules. *Information and Control*, 12:415–425, 1968.
26. A. Gabrielian. *Pure Grammars and Pure Languages.* Research Report CSRR 2027, Department of Applied Analysis and Computer Science, University of Waterloo, Ontario, Canada, 1970.
27. S. Ginsburg and E. H. Spanier. Control sets on grammars. *Mathematical Systems Theory*, 2(2):159–177, 1968.
28. S. A. Greibach and J. E. Hopcroft. Scattered context grammars. *Journal of Computer and System Sciences*, 3(3):233–247, 1969.
29. M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Boston, 1978.
30. G. T. Herman and G. Rozenberg. *Developmental Systems and Languages.* North-Holland Publishing Company, Amsterdam, 1975.
31. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Boston, 3rd edition, 2006.

32. G. Horváth and A. Meduna. On state grammars. *Acta Cybernetica*, 1988(8):237–245, 1988.
33. O. H. Ibarra. Simple matrix languages. *Information and Control*, 17:359–394, 1970.
34. L. Kari, A. Mateescu, G. Păun, and A. Salomaa. Teams in cooperating grammar systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 7:347–359, 1995.
35. J. Kari and L. Sântean. The impact of the number of cooperating grammars on the generative power. *Theoretical Computer Science*, 98:621–633, 1992.
36. T. Kasai. An hierarchy between context-free and context-sensitive languages. *Journal of Computer and System Sciences*, 4:492–508, 1970.
37. H. C. M. Kleijn and G. Rozenberg. Context-free like restrictions on selective rewriting. *Theoretical Computer Science*, 16:237–269, 1981.
38. H. C. M. Kleijn and G. Rozenberg. Sequential, continuous, and parallel grammars. *Information and Control*, 48:221–260, 1981.
39. H. C. M. Kleijn and G. Rozenberg. Multi grammars. *International Journal of Computer Mathematics*, 12:177–201, 1983.
40. R. Kocman, Z. Křivka, and A. Meduna. On double-jumping finite automata. In *Proceedings of Eight Workshop on Non-Classical Models of Automata and Applications (NCMA 2016)*, pages 195–210, 2016.
41. R. Kocman, Z. Křivka, A. Meduna, and B. Nagy. A jumping 5' $\rightarrow$ 3' Watson-Crick finite automata model. In *Tenth Workshop on Non-Classical Models of Automata and Applications (NCMA 2018)*, pages 117–132, 2018.
42. R. Kocman and A. Meduna. On parallel versions of jumping finite automata. In *Proceedings of the 2015 Federated Conference on Software Development and Object Technologies (SDOT 2015)*, volume 511 of *Advances in Intelligent Systems and Computing*, pages 142–149. Springer International Publishing, 2016.
43. D. Kolář and A. Meduna. Regulated pushdown automata. *Acta Cybernetica*, 2000(4):653–664, 2000.
44. D. Kolář and A. Meduna. One-turn regulated pushdown automata and their reduction. *Fundamenta Informaticae*, 51(4):399–405, 2002.
45. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, Massachusetts, 1992.
46. K. Krithivasan and R. Siromoney. Parallel context-free grammar. *Information and Control*, 24:155–162, 1974.
47. Z. Křivka. *Rewriting Systems with Restricted Configurations.* Faculty of Information Technology, Brno University of Technology, Brno, 2008.
48. Z. Křivka, J. Kučera, and A. Meduna. On k#$-rewriting systems. *Romanian Journal of Information Science and Technology (ROMJIST)*, 21(3):278–287, 2018.
49. Z. Křivka, J. Kučera, and A. Meduna. Jumping pure grammars. *The Computer Journal*, 62(1):30–41, 2019.
50. Z. Křivka and A. Meduna. General top-down parsers based on deep pushdown expansions. In *Proceedings of 1st International Workshop on Formal Models (WFM '06)*, pages 11–18, 2006.

51. Z. Křivka and A. Meduna. Jumping grammars. *International Journal of Foundations of Computer Science*, 26(6):709–731, 2015.
52. Z. Křivka, A. Meduna, and R. Schönecker. Generation of languages by rewriting systems that resemble automata. *International Journal of Foundations of Computer Science*, 17(5):1223–1229, 2006.
53. J. Kučera and A. Meduna. On state-synchronized automata systems. *Schedae Informaticae*, 2015(24):221–237, 2016.
54. J. Kučera, A. Meduna, and O. Soukup. Absolutely unlimited deep pushdown automata. In J. Kofroň and T. Vojnar, editors, *Proceedings of the 10th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2015)*, pages 36–44, 2015.
55. M. K. Levitina. On some grammars with global productions. *NTI*, 2(3):32–36, 1972.
56. A. Lindenmayer. Mathematical models for cellular interaction in development I. *Journal of Theoretical Biology*, 18(3):280–299, 1968.
57. A. Lindenmayer. Mathematical models for cellular interaction in development II. *Journal of Theoretical Biology*, 18(3):300–315, 1968.
58. R. Lukáš. *Multigenerativní Gramatické Systémy.* PhD thesis, Faculty of Information Technology, Brno University of Technology, Brno, 2006.
59. R. Lukáš and A. Meduna. Multigenerative grammar systems and matrix grammars. *Kybernetika*, 46(1):68–82, 2010.
60. C. Martín-Vide, A. Mateescu, and V. Mitrana. Parallel finite automata systems communicating by states. *International Journal of Foundations of Computer Science*, 13(5):733–749, 2002.
61. T. Masopust. *Formal Models: Regulation and Reduction.* Faculty of Information Technology, Brno University of Technology, Brno, 2007.
62. T. Masopust and A. Meduna. Self-regulating finite automata. *Acta Cybernetica*, 18:135–153, 2007.
63. H. A. Maurer, A. Salomaa, and D. Wood. Pure grammars. *Information and Control*, 44:47–72, 1980.
64. A. Meduna. A trivial method of characterizing the family of recursively enumerable languages by scattered context grammars. *Bulletin EATCS*, 56:104–106, 1995.
65. A. Meduna. Syntactic complexity of scattered context grammars. *Acta Informatica*, 32:285–298, 1995.
66. A. Meduna. *Automata and Languages: Theory and Applications.* Springer, London, 2000.
67. A. Meduna. Simultaneously one-turn two-pushdown automata. *International Journal of Computer Mathematics*, 80:679–687, 2003.
68. A. Meduna. Deep pushdown automata. *Acta Informatica*, 2006(98):114–124, 2006.
69. A. Meduna and O. Soukup. Jumping scattered context grammars. *Fundamenta Informaticae*, 152:51–86, 2017.
70. A. Meduna and P. Zemek. One-sided random context grammars. *Acta Informatica*, 48(3):149–163, 2011.

71. A. Meduna and P. Zemek. Jumping finite automata. *International Journal of Foundations of Computer Science*, 23(7):1555–1578, 2012.
72. A. Meduna and P. Zemek. *Regulated Grammars and Automata.* Springer, New York, 2014.
73. V. Mihalache. Matrix grammars versus parallel communicating grammar systems. In G. Păun, editor, *Mathematical Aspects of Natural and Formal Languages*, volume 43 of *World Scientific Series in Computer Science*, pages 293–318. World Scientific, Singapore, 1994.
74. V. Mitrana. Hybrid cooperating distributed grammar systems. *Computers and Artificial Intelligence*, 2:83–88, 1993.
75. R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.
76. G. Păun. A new generative device: valence grammars. *Revue Roumaine de Mathématiques Pures et Appliquées*, 25:911–924, 1980.
77. G. Păun. On the generative capacity of hybrid CD grammar systems. *Journal of Information Processing and Cybernetics*, 30(4):231–244, 1994.
78. G. Păun and L. Sântean. Parallel communicating grammar systems: the regular case. *Analele Universităţii din Bucureşti, Seria Matematică-Informatică*, 1989(2):55–63, 1989.
79. R. D. Rosebrugh. *Restricted Parallelism and Regular Grammars.* Master's thesis, McMaster University, Hamilton, Ontario, 1972.
80. R. D. Rosebrugh and D. Wood. Restricted parallelism and right linear grammars. *Utilitas Mathematica*, 7:151–186, 1975.
81. A. L. Rosenberg. On multi-head finite automata. *IBM Journal of Research and Development*, 10(5):388–394, 1966.
82. D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the ACM*, 16(1):107–131, 1969.
83. G. Rozenberg. Extension of tabled 0L systems and languages. *International Journal of Computer and Information Sciences*, 2:311–334, 1973.
84. G. Rozenberg. T0L systems and languages. *Information and Control*, 23:262–283, 1973.
85. G. Rozenberg and P. G. Doucet. On 0L-languages. *Information and Control*, 19:302–318, 1971.
86. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages: Beyond Words.* Springer-Verlag, Berlin, 1997.
87. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages: Linear Modeling: Background and Application.* Springer-Verlag, Berlin, 1997.
88. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages: Word, Language, Grammar.* Springer-Verlag, Berlin, 1997.
89. A. Salomaa. *Formal Languages.* Academic Press, London, 1973.
90. A. Salomaa. *Computation And Automata.* Cambridge University Press, Cambridge, 1985.
91. L. Sântean. Parallel communicating grammar systems. *Current Trends in Theoretical Computer Science: Essays and Tutorials*, 40:603–615, 1993.

92. D. A. Simovici and R. L. Tenney. *Theory of Formal Languages with Applications.* World Scientific, Singapore, 1999.

93. R. Siromoney and K. G. Subramanian. Selective substitution array grammars. *Information Sciences*, 25:73–83, 1981.

94. I. H. Sudborough. On tape-bounded complexity classes and multi-head finite automata. *Journal of Computer and System Sciences*, 10(1):62–76, 1975.

95. A. P. J. van der Walt. Random context grammars. In *Proceedings of Symposium on Formal Languages*, pages 163–165, 1970.

96. G. Winskel and M. Nielsen. Models for concurrency. In *Handbook of Logic and the Foundations of Computer Science*, volume 4, pages 1–148. Oxford University Press, 1995.

97. D. Wood. m-parallel n-right linear simple matrix languages. *Utilitas Mathematica*, 8:3–28, 1975.

98. G. Zetzsche. On erasing productions in random context grammars. In *ICALP '10: Proceedings of the 37th International Colloquium on Automata, Languages, and Programming*, volume 6199 of *Lecture Notes in Computer Science*, pages 175–186. Springer-Verlag, Berlin, Heidelberg, 2010.

Notation Index

Term	Page	Meaning
$\emptyset$	7	empty set
$\vdash_{\mathrm{M}}$	14	M-based relation of direct move
$\Rightarrow_{\mathrm{G}}$	12	G-based relation of direct derivation
$\{0, 1, \ldots, 7\}$	7	a set containing digits from 0 to 7
2^{P}	7	power set of P
${}^{a}\vdash_{\mathrm{M}}$	51	M-based relation of direct move (AUDPDA)
${}^{a}_{e}\vdash_{\mathrm{M}}$	51	M-based relation of direct expansion move (AUDPDA)
${}^{a}_{p}\vdash_{\mathrm{M}}$	51	M-based relation of direct pop move (AUDPDA)
$\{a \mid \pi(a)\}$	7	a set containing all elements that fulfill some property π
$\mathrm{A}_1 \times \mathrm{A}_2$	8	Cartesian product of A_1 and A_2
$a \in \mathrm{P}$	7	a is a member of P
$a \notin \mathrm{P}$	7	a is not a member of P
$(a_1, a_2, \ldots, a_n)$	8	n-tuple
$\{a, b, c\}$	7	a set containing elements a, b, and c
$\mathrm{alph}(w)$	10	the set of all symbols that appear in w
$\mathrm{card}(\mathrm{P})$	7	cardinality of P
$\mathrm{C}_n(\mathrm{A})$	8	Cartesian product of n sets A
$\mathrm{depth}(\mathrm{M})$	25	a depth of DPDA M
$\mathrm{d}_\ell(u, v)$	104	Levenshtein distance between u and v
$\mathscr{D}_n$	13	Dyck language of index n
$\mathrm{domain}(\rho)$	8	domain of ρ
ε	9	empty word
${}_{e}\vdash_{\mathrm{M}}$	26	M-based relation of direct expansion move (DPDA)
$\mathscr{G}(\mathrm{X})$	23	a set of grammars of type X
${}_{h}\Rightarrow_{\mathrm{G}}$	29	G-based relation of h-mode direct derivation
id_{A}	8	identity relation over A
$\mathrm{Ind}(\mathrm{G})$	23	the index of a grammar G

Term	Page	Meaning
$p\rho q$	8	$(p, q) \in \rho$
ρ^{+}	9	transitive closure of ρ
ρ^{*}	9	reflexive and transitive closure of ρ
ρ^{k}	8	k-fold product of ρ
$\rho \circ \sigma$	8	composition of ρ and σ
${}^{r}\vdash_{\mathrm{M}}$	58	M-based relation of direct move (RUDPDA)
${}^{r}_{e}\vdash_{\mathrm{M}}$	58	M-based relation of direct expansion move (RUDPDA)
${}^{r}_{p}\vdash_{\mathrm{M}}$	57	M-based relation of direct pop move (RUDPDA)
$r : x \to y$	12	a rule $x \to y$ has a label r
range(ρ)	8	range of ρ
$\mathcal{R}_{\Gamma}$	39	Cartesian product of sets of rules of components of Γ
rhs(r)	12	right-hand side of a rule r
${}_{rj}\Rightarrow_{\mathrm{G}}$	29	G-based relation of right jump direct derivation
Σ^{+}	9	a set of all non-empty words over Σ
Σ^{*}	9	a set of all words over Σ
	10	the universal language over Σ
${}_{s}\Rightarrow_{\mathrm{G}}$	29	G-based relation of sequential direct derivation
subword(w)	10	the set of all subwords of w
suffix(w)	10	the set of all suffixes of w
sup S	9	supremum of S
${}_{sw}\Rightarrow_{\mathrm{G}}$	104	G-based relation of swapping direct derivation
$\mathbb{U}$	7	universe
$u_0 \Rightarrow^{n}_{\mathrm{G}} u_n\ [\pi_n]$	12	u_n is derived from u_0 by consecutive application of sequence of rules π_n
$\phi(\mathrm{L})$	11	commutative (Parikh) map of L
$\phi(w)$	11	commutative (Parikh) image of w
$\phi(x) = y$	9	$(x, y) \in \phi$
$\chi_0 \vdash^{n}_{\mathrm{M}} \chi_n\ [\pi_n]$	14	a move from χ_0 to χ_n performed by consecutive application of sequence of rules π_n
$\lvert x \rvert$	9	length of word x
x^{i}	9	ith power of word x
$x \to y$	11	(x, y)
xy	9	concatenation of words x and y
Ψ_{f}	39	control language of SCAS extended about words formed from states from its final configurations

Language Family Index

Family	Page	Description
0L	31	family of languages generated by 0L systems
$\mathbf{0L}^{-\varepsilon}$	31	family of languages generated by propagating 0L systems
AUDPDA	51	family of languages accepted by absolutely unlimited deep pushdown automata
$\mathbf{AUDPDA}^{-\varepsilon}$	51	family of languages accepted by propagating absolutely unlimited deep pushdown automata
CF	13	context-free languages
CS	13	context-sensitive languages
$\mathbf{DPDA}_k$	26	family of languages accepted by deep pushdown automata of depth k
$\mathbf{DPDA}_k^{\varepsilon}$	26	family of languages accepted by deep pushdown automata of depth k by empty pushdown
$_{\text{fin}}\mathbf{X}$	23	family of languages of finite index generated by grammars of type X
JPP	65	family of languages generated by pure grammars in jumping parallel mode
$\mathbf{JPP}^{-\varepsilon}$	65	family of languages generated by propagating pure grammars in jumping parallel mode
JPPCF	66	family of languages generated by pure context-free grammars in jumping parallel mode
$\mathbf{JPPCF}^{-\varepsilon}$	66	family of languages generated by propagating pure context-free grammars in jumping parallel mode
JSP	65	family of languages generated by pure grammars in jumping sequential mode
$\mathbf{JSP}^{-\varepsilon}$	65	family of languages generated by propagating pure grammars in jumping sequential mode
JSPCF	66	family of languages generated by pure context-free grammars in jumping sequential mode
$\mathbf{JSPCF}^{-\varepsilon}$	66	family of languages generated by propagating pure context-free grammars in jumping sequential mode

Family	Page	Description
$\mathscr{L}(\text{1t-PDA})_\varepsilon$	16	family of languages accepted by one-turn pushdown automata by empty pushdown
$\mathscr{L}(\text{1t-PDA})_f$	16	family of languages accepted by one-turn pushdown automata by final state
$\mathscr{L}(\text{1t-PDA})_{f\varepsilon}$	16	family of languages accepted by one-turn pushdown automata by final state and empty pushdown
$\mathscr{L}(\text{2PDA})_\varepsilon$	17	family of languages accepted by two-pushdown automata by empty pushdown
$\mathscr{L}(\text{2PDA})_f$	17	family of languages accepted by two-pushdown automata by final state
$\mathscr{L}(\text{2PDA})_{f\varepsilon}$	17	family of languages accepted by two-pushdown automata by final state and empty pushdown
$\mathscr{L}(\text{d2PDA})_\varepsilon$	17	family of languages accepted by deterministic two-pushdown automata by empty pushdown
$\mathscr{L}(\text{d2PDA})_f$	17	family of languages accepted by deterministic two-pushdown automata by final state
$\mathscr{L}(\text{d2PDA})_{f\varepsilon}$	17	family of languages accepted by deterministic two-pushdown automata by final state and empty pushdown
$\mathscr{L}(\text{dFA})$	14	family of languages accepted by deterministic finite automata
$\mathscr{L}(\text{dPDA})_\varepsilon$	16	family of languages accepted by deterministic pushdown automata by empty pushdown
$\mathscr{L}(\text{dPDA})_f$	16	family of languages accepted by deterministic pushdown automata by final state
$\mathscr{L}(\text{dPDA})_{f\varepsilon}$	16	family of languages accepted by deterministic pushdown automata by final state and empty pushdown
$\mathscr{L}(\text{FA})$	14	family of languages accepted by finite automata
LIN	13	linear languages
$\mathscr{L}_k(\#\$\text{RS})$	82	family of languages generated by $k\#\$$-rewriting systems
$\mathscr{L}_k(\#\text{RS})$	29	family of languages generated by #-rewriting systems of index k
$\mathscr{L}(\text{PDA})_\varepsilon$	15	family of languages accepted by pushdown automata by empty pushdown
$\mathscr{L}(\text{PDA})_f$	15	family of languages accepted by pushdown automata by final state
$\mathscr{L}(\text{PDA})_{f\varepsilon}$	15	family of languages accepted by pushdown automata by final state and empty pushdown
$\mathscr{L}_{\text{pfx}}$	10	prefix languages
$\mathscr{L}(\text{s1t-2PDA})_\varepsilon$	18	family of languages accepted by simultaneously one-turn two-pushdown automata by empty pushdown
$\mathscr{L}(\text{s1t-2PDA})_f$	18	family of languages accepted by simultaneously one-turn two-pushdown automata by final state
$\mathscr{L}(\text{s1t-2PDA})_{f\varepsilon}$	18	family of languages accepted by simultaneously one-turn two-pushdown automata by final state and empty pushdown

Subject Index