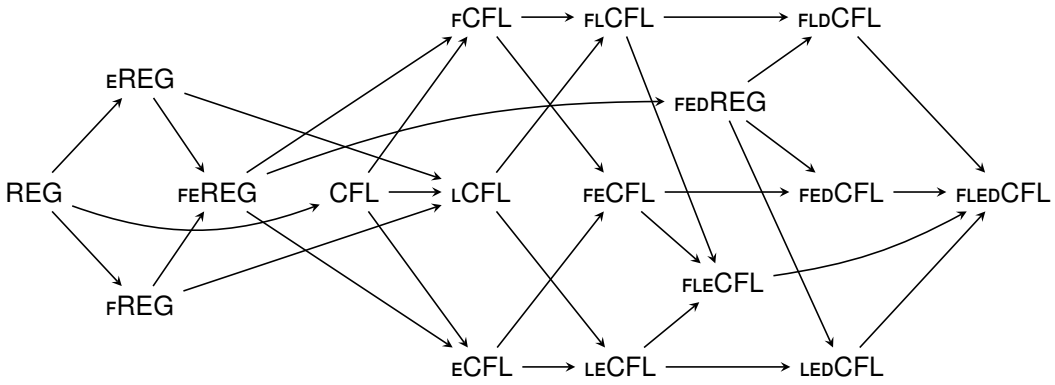
Position-and-Length-Dependent Context-Free Grammars – A New Type of Restricted Rewriting



Vom Fachbereich Informatik der
Technischen Universität Kaiserslautern
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat)
genehmigte Dissertation

von Frank Weinberg

Dekan:	Prof. Dr. Klaus Schneider
Vorsitzender der Prüfungskommission:	Prof. Dr. Hans Hagen
1. Berichterstatter:	Prof. Dr. Markus Nebel
2. Berichterstatter:	Prof. Dr. Hening Fernau

Datum der wissenschaftlichen Aussprache: 3. März 2014

D 386

Abstract

For many decades, the search for language classes that extend the context-free laguages enough to include various languages that arise in practice, while still keeping as many of the useful properties that context-free grammars have – most notably cubic parsing time – has been one of the major areas of research in formal language theory. In this thesis we add a new family of classes to this field, namely position-and-length-dependent context-free grammars. Our classes use the approach of regulated rewriting, where derivations in a context-free base grammar are allowed or forbidden based on, e.g., the sequence of rules used in a derivation or the sentential forms, each rule is applied to. For our new classes we look at the yield of each rule application, i.e. the subword of the final word that eventually is derived from the symbols introduced by the rule application. The position and length of the yield in the final word define the position and length of the rule application and each rule is associated a set of positions and lengths where it is allowed to be applied.

We show that – unless the sets of allowed positions and lengths are erally complex – the languages in our classes can be parsed in the same time as context-free grammars, using slight adaptations of well-known parsing algorithms. We also show that they form a proper hierarchy above the context-free languages and examine their relation to language classes defined by other types of regulated rewriting.

We complete the treatment of the language classes by introducing pushdown automata with position counter, an extension of traditional pushdown automata that recognizes the languages generated by position-and-length-dependent context-free grammars, and we examine various closure and decidability properties of our classes. Additionally, we gather the corresponding results for the subclasses that use right-linear resp. left-linear base grammars and the corresponding class of automata, finite automata with position counter.

Finally, as an application of our idea, we introduce length-dependent stochastic context-free grammars and show how they can be employed to improve the quality of predictions for RNA secondary structures.

Acknowledgements

Of course, producing such a thesis is usually not possible without the support of various people. In the case of this thesis, the major supporters are the Algorithms and Complexity Group of Prof. Markus Nebel. They did not only provide a very pleasant environment to work in – so pleasant that it actually proved difficult to cut myself loose from it – but they also allowed many helpful discussions, not only on the topics of the thesis, helping to find or evaluate new ideas, but also on other topics, helping to free the mind from failed approaches.

A special “Thank You!” goes to Raphael Reitzig and Anne Berres, both of whom made numerous helpful suggestions when proofreading – or in some places rather fighting through – earlier drafts of this document.

Contents

1. Introduction

In this thesis, we will introduce position-and-length-dependent context-free grammars, a new type of formal grammars based on the abstract idea of restricted rewriting.

The basic idea of restricted rewriting is to extend the generative power of a simple grammar class – typically the context-free grammars – by restricting the applicability of the rules based on conditions that may refer to things as the current sentential form or the sequence of rules applied before. As an example, consider context-sensitive grammars. In these, each rule replaces a single nonterminal symbol, just as the rules of a context-free grammar do. But in the context-sensitive grammar the rule may only be applied if the replaced nonterminal is surrounded by a specific context that is given with the rule.

In this example the condition on a rule is local, i.e., we can decide if a rule is applicable to a specific nonterminal in a sentential form by looking at the sentential form alone. Other types of restricted rewriting have global constraints. For example, in unordered vector grammars, the rules are grouped into sets, called vectors, and a derivation is valid, if all the rules in a vector have been used equally often in the derivation. (The number of applications may differ between vectors.)

For position-and-length-dependent context-free grammars we use a mixed approach. While the validity is checked for each individual rule application, the conditions depend on the derivation as a whole and thus can only be verified for a complete derivation. Specifically, we look at the *yield* of a rule application, i.e. the subword that is eventually derived from the symbols introduced by the rule. We then allow or forbid the rule application based on position and length of this yield, where the position is defined by the number of characters in front of resp. behind the yield.

1.1. Overview

The remaining sections of this chapter introduce – after a short explanation of some notational conventions we use – the motivation that prompted the research of restricted rewriting along with several of the approaches that have been previously examined. Then, we give an informal overview of position-and-length-dependent grammars and the subclasses we are going to examine. These subclasses result from allowing fewer restrictions or using right- resp. left-linear base grammars.

We formally introduce the grammar classes in Chapter 2. There we also give some basic results, e.g. the fact that our grammars can be transformed into equivalent ones in Chomsky Normal Form.

In Chapter 3, we define equivalent automata for each of the classes. Unsurprisingly, they result from adding position checks and/or length checks to pushdown automata resp. finite automata in a suitable way. We also show that the same language classes result from right- and left-linear grammars, though the type of restrictions needed to arrive at some class can differ for the two grammar types.

In Chapter 4, we describe how several well-known parsing algorithms can be adapted to the new grammar classes. This can be done without using additional space or time except what is necessary to verify membership in the restriction sets for triples encountered during the parsing. This is not immediately obvious since, other than for traditional CFG, the conditions we introduce are not locally decidable at each step of the derivation as they require information about how large the yield of the rule and the surrounding derivation will be.

We then go on in Chapter 5 to show that all the classes not yet shown to coincide are actually distinct. We also show that they do not coincide with any of the previously known classes presented in the introduction. Furthermore, we examine the effect that the complexity of restriction sets has on the complexity of generated languages. The chapter concludes with a result that shows that position and length dependent grammars can significantly improve the description complexity as compared to their traditional counterparts.

Chapters 6 and 7 conclude the examination of the language classes we define. Here, we examine decidability resp. closure properties of each language class. For decidability we can only report negative results, while at least some of the closure properties of context-free resp. regular languages carry over.

Finally, in Chapter 8, we introduce stochastic length-dependent context-free grammars and show that they can be used to improve the prediction quality of known algorithms based on stochastic context-free grammars that predict the secondary structure of RNA molecules.

1.2. Notation

We asssume familiarity with the basic notions of regular and context-free grammars as well as their corresponding accepting automata. An introduction can be found in [Neb12] (in german) or [Har78].

In this section we only explain some less common notations we use. For an exhaustive list, refer to Appendix A.

First, we introduce a shorthand for sets of natural numbers resp. tuples of those.
For $M_1, M_2 \subseteq \mathbb{N}^i$ and $\circ$ an operation on $\mathbb{N}^i$, $M_1 \circ M_2 = \{m_1 \circ m_2 \in \mathbb{N}^i \mid m_1 \in M_1,\ m_2 \in M_2\}$. So, e.g., $\{2\} \cdot \mathbb{N}$ denotes the set of even numbers, while $\{2\} \cdot \mathbb{N} + \{1\}$ denotes the odd numbers. Be aware of the difference between $\mathbb{N}^2$, the set of pairs of natural numbers, and $\mathbb{N}^{\{2\}}$, the set of squares.

For any sets, $\sqcup$ denotes disjoint union, i.e. we assume w.l.o.g. that the sets united are disjoint.

If Σ is an alphabet, Σ_ϵ denotes $\Sigma \cup \{\epsilon\}$, where ϵ denotes the empty word. For a string α, α_i denotes the i-th character of α and $\alpha_{i\ldots j}$ denotes $\alpha_i \cdot \cdots \cdot \alpha_j$, where $\cdot$ denotes concatenation. α^R

If ωG denotes a class of grammars, the corresponding class of languages is denoted by ωL and vice versa. We denote the regular languages by REG, right-linear resp. left-linear grammars by RLING resp. LLING, the context-free languages by CFL, the context-sensitive languages by CSL, the decidable languages by DEC and the recursive enumerable languages by RE.

If C and pC are two classes of grammars or languages we will write [p]C to indicate that a statement holds for both of them. If a statement includes multiple classes with the same bracketed prefix, the statement is only meant to hold for those classes where the presence of the prefix coincides. So [p]C = [p]D means pC = pD and C = D, but *not* pC = D or C = pD. If multiple different bracketed prefixes are present in one statement they are understood to be indepent of each other. So the equation [p]C = [q]D = [p][q]E implies the four equalities pC = qD = pqE, pC = D = pE, C = qD = qE and C = D = E.

In order to avoid confusion with derivations, we use $\rightsquigarrow$ to denote implication and $\leftrightsquigarrow$ to denote equivalence. When a result or definition is taken from literature without modifications, except possibly syntactical changes, we denote the source directly at the name of the definition, theorem, etc. If a proof is an adaptation of the respective proof for established grammar classes, we state this either directly in the proof or in the surrounding text.

1.3. Motivation

When Chomsky introduced his hierarchy of formal language classes in [Cho59], one of the problems that people very soon started to consider was this: Where in this hierarchy do the languages that arise in practice belong. Most prominently, this included natural languages and programming languages. But languages arising in other fields later on were also examined, e.g. languages modeling the folding of RNA molecules in bioinformatics.

As it turned out many languages of practical interest including all examples mentioned in the previous paragraph are context-sensitive but not context-free [Shi87], [Flo62], [Sea92]. This is unsatisfying since the class of context-sensitive grammars is missing some of the desirable properties of context-free grammars. Most prominently the word problem for the former is PSPACE-complete [GJ79], while it is solvable in cubic time for the latter [Ear70].

Thus, people started to look for extensions of context-free grammars that include the languages they were interested in, while still keeping as many of the useful features that context-free languages have, especially

polynomial parsing time.

In the case of programing languages, it eventually turned out that most necessary features can be modeled by deterministic context-free grammars, the exception being long-distance constraints like the requirement that every used function or variable is declared somewhere and similar restrictions. Here, the pragmatic solution was to remove these constraints from the formal syntax and instead enforce them at a different stage of the compilation or even at runtime.

For natural languages, the definition of *mildly context-sensitive languages* in [Jos85] has become widely accepted as being a good choice for a class that captures natural languages without being too general. The class is described by the following properties:

1. It contains all context-free languages.
2. It allows for a limited amount of cross-serial dependencies (e.g. the language $\{ww \mid w \in \Sigma^*\}$ should be included, while the language where each word consists of the same number of a's b's and c's in arbitrary order should not).
3. Its languages can be parsed in polynomial time and
4. they are semilinear.[1]

A significant number of the formalisms suggested can, on an abstract level, be viewed as consisting of a context-free grammar and a set of restrictions that constrain, which of the rules may be applied at which point of the derivation. The restrictions may be based on things like the rules applied previously or the symbols present in the current sentential form.

We will give an overview of several of the approaches that have been previously suggested in this vein, before introducing position and length restrictions, the new type of restrictions we are going to examine in this thesis.

1.4. Known Forms of Restricted Rewriting

1.4.1. Conditions on Sentential Forms

The first kind of restrictions we consider are restrictions based on the sentential form to which a (context-free) rule is applied. This is similar to context-sensitive grammars, the difference being that for context-sensitive grammars only the immediate surroundings of the replaced nonterminal are considered, while for the following grammars we look at the whole sentential form.

The first class of this kind were *conditional grammars* ([Fri68]). In these grammars each rule is associated with a regular set over the terminal and nonterminal symbols and the rule may only be applied to sentential forms that are in this set. We call the resulting grammar classes $[\epsilon]$CG, where the presence resp. absence of the prefix ϵ indicates throughout this chapter that ϵ-rules are allowed resp. forbidden.

As it turns out, CL = CSL $\subsetneq$ ϵCL = RE ([Fri68], [Pău79]).

The restriction was sharpened for *semi-conditional grammars* ([ϵ]sCG; [Kel84], [Pă85]). Here, each rule is accompanied by two strings over the terminals and nonterminals, called the *permitted context* and *forbidden context* respectively. A rule may be applied to a sentential form only if the permitted context appears as a substring of the sentential form, while the forbidden context does not.

Again, we find sCL = CSL $\subsetneq$ ϵsCL = RE, even when we further restrict the grammar such that for each rule at least one of the contexts is empty, each permitted context is of length at most 2, each forbidden context is of length 1, and the contexts are the same for rules with the same left-hand side ([DM12]).

Little is known about the subclasses that allow only one type of context globally ([Mas09], [DM12]).

As a final grammar type of this kind, we consider *random context grammars* ([ϵ]RCG; [VdW70]). Just like semi-conditional grammars, they have a permitted and a forbidden context for each rule, but this time the

[1] A language L over $\{a_1, \ldots, a_n\}$ is semilinear, iff its set of Parikh vectors $\{(|w|_{a_1}, \ldots, |w|_{a_n}) \mid w \in L\}$ is a semilinear set, i.e. a finite union of linear sets.

contexts consist of a set of nonterminals. The rule may be applied if all symbols of the permitted context and no symbol from the forbidden context are present.

In contrast to sCG, the subclasses with only one type of contexts are also thoroughly studied. They are called *permitting* resp. *forbidding* random context grammars ([ϵ]PRCG, [ϵ]FRCG).

We have

$$\begin{aligned}
&\text{CFL} \subsetneq \text{PRCL} = \epsilon\text{PRCL} \subsetneq \text{RCL} \subsetneq \text{CSL} && \text{([May72], [Zet10], [Fer96], [Ros69]),}\\
&\text{CFL} \subsetneq \text{FRCL} \subseteq \epsilon\text{FRCL} \subsetneq \text{DEC} \subsetneq \epsilon\text{RCL} = \text{RE} && \text{([May72], [BF94]),}\\
&\text{FRCL} \subsetneq \text{RCL} && \text{([vdWE00]).}
\end{aligned}$$

With RCL, FRCL, and PRCL this gives us the first classes between the context-free and context-sensitive languages. However, the membership problem for FRCL and RCL is NP-hard ([BF94]), and to the best of the author's knowledge, no polynomial time algorithm for the membership problem of PRCL is known, either.

1.4.2. Restrictions on the Rule Sequence

A different approach is to consider each derivation as a sequence of rule applications, and allow only some of these sequences.

In this setting it can be beneficial to allow a rule to be applied to a sentential form that does not contain the left-hand side of the rule in order to satisfy constraints on the rule sequence. This is called applying the rule in *appearance checking* mode, and it is defined to leave the sentential form unaltered. Usually, application in appearance checking mode is only allowed for a subset of the rules which is declared in the grammar. For the following types of grammars we will denote the classes resulting from allowing the declaration of such a subset with the prefix letter A. If no such set may be declared, we omit the A.

The simplest variant of this kind of restriction is that of *programmed grammars* ([ϵ][A]PG; [Ros69]). Here, each rule is equipped with a list of rules that may be applied immediately after the rule in question.

For these classes, the following inclusions hold:

$$\begin{aligned}
&\text{PRCL} \subseteq \text{PL} \subsetneq \text{APL} = \text{RCL} && \text{([May72], [HJ94]),}\\
&\text{PL} \subseteq \epsilon\text{PL} \subsetneq \epsilon\text{APL} = \text{RE} && \text{([GW89], [Ros69]),}\\
&\text{APL} \not\subset \epsilon\text{PL} && \text{([GW89]),}\\
&\text{PRCL} \subsetneq \epsilon\text{PL} && \text{([Zet10]).}
\end{aligned}$$

Again, we have found classes between the context-free and context-sensitive languages, but as for the random context grammars, the membership problem is NP-hard ([GW89]).

Extending the range of each constraint, we get to *matrix grammars* ([ϵ][A]MG; [Abr65]). Here, rules are grouped into so-called matrices. We always have to apply the rules of a matrix in direct sequence and in the order given in the grammar. Allowing them to be applied in arbitrary order gives *unordered matrix grammars* ([ϵ][A]UMG; [CM73]). Neither of these concepts changes the generative power compared to programmed grammars ([Sal70], [CM73]), however:

$$[\epsilon][\text{A}]\text{UML} = [\epsilon][\text{A}]\text{ML} = [\epsilon][\text{A}]\text{PL}.$$

A variation are *scattered context grammars* ([ϵ][A]SCG; [GH69]). Here, we apply all the rules of a matrix in parallel and also require the replaced nonterminals to appear in the left-to-right order implied by the order of rules in the matrix. If the latter requirement is dropped, we get *unordered scattered context grammars* ([ϵ][A]USCG; [MR71]).

The resulting language classes fit into the hierarchy as follows:

$$\begin{aligned}
&[\epsilon][\text{A}]\text{USCL} = [\epsilon][\text{A}]\text{PL} && \text{([May72]),}\\
&\text{PL} \subsetneq \text{SCL} \subseteq \text{ASC} = \text{CSL} && \text{([GW89], [Cre73]),}\\
&\epsilon\text{SCL} = \epsilon\text{ASC} = \text{RE} && \text{([May72]).}
\end{aligned}$$

From this, we already see that the membership problem for the new class SCL is NP-hard as well.

Another variation of matrix grammars are *vector grammars* ([ϵ][A]VG; [CM73]). Here, we relax the requirement that the rules in a matrix or vector, as they are now called, are applied in immediate succession.

Instead we may interleave the application of multiple vectors (including multiple copies of the same vector). If we further loosen the restriction so that the rules of a vector may be applied in arbitrary order we get *unordered vector grammars* ([ϵ][A]UVG) ([CM73]).

For the resulting language classes we find the following relations:

$$\begin{array}{ll} \mathrm{VL} = \epsilon\mathrm{VL} = \epsilon\mathrm{PL} \subsetneq \mathrm{AVL} \subsetneq \mathrm{DEC} \subsetneq \epsilon\mathrm{AVL} = \mathrm{RE} & ([\text{Zet11a}], [\text{CM73}], [\text{GW89}], [\text{WZ}]), \\ \mathrm{APL} \subseteq \mathrm{AVL} & ([\text{CM73}]), \\ \mathrm{CFL} \subsetneq \mathrm{UVL} = \epsilon\mathrm{UVL} = \mathrm{AUVL} = \epsilon\mathrm{AUVL} \subsetneq \mathrm{PL} & ([\text{FS02}], [\text{CM74}], [\text{WZ}]). \end{array}$$

This implies two new classes AVL and UVL. While for AVL the NP-hardness of the membership problem is again immediate, UVL behaves friendlier. For this class, membership can be decided in polynomial time ([Sat96]), and it contains only semilinear languages ([CM74]). However, the class is not mildly context-sensitive since $\{ww \mid w \in \Sigma^*\} \notin \mathrm{UVL}$.

While the previously considered grammars enforce a local structure on the rule sequence the *regularly controlled grammars* ([ϵ][A]RCG; [GS68], [Sal69]) consider the sequence as a whole and require it to be in a regular set that is given with the grammar. Again, we get some classes we already know ([Sal70]):

$$[\epsilon][\mathrm{A}]\mathrm{RCL} = [\epsilon][\mathrm{A}]\mathrm{PL}.$$

Another variation are *petri net grammars* ([ϵ][A]PNG; [DT09]). Here, the regular set is replaced by a petri net and a mapping between grammar rules and petri net transitions. It is then required that the sequence of rules corresponds to a valid sequence of transitions of the petri net.

Again, we get no new classes ([Zet11b], [DT09], [WZ]):

$$[\epsilon][\mathrm{A}]\mathrm{PNL} = [\epsilon][\mathrm{A}]\mathrm{VL}.$$

The final type are *valence grammars* ([Pau80]). Here each rule is associated with an element from a monoid. A derivation, when viewed as a sequence of rule applications, then yields a sequence of monoid elements, and combining these elements under the operation of the monoid yields a value. The derivation is valid if and only if that value is the neutral element. Obviously, different monoids define different grammar classes.

The monoid $(\mathbb{Z}, +, 0)$ defines the *additive valence grammars* ($\mathbb{Z}$VG). $(\mathbb{Q}_+, \cdot, 1)$ defines the *multiplicative valence grammars* ($\mathbb{Q}$VG). For each $i \in \mathbb{N}$, $\mathbb{Z}^i$VG resp. $\mathbb{Q}^i$VG denotes the class resulting from using the monoid $(\mathbb{Z}^i, +, 0)$ resp. $(\mathbb{Q}_+^i, \cdot, 0)$. For all these monoids allowing ϵ-rules or appearance checking does not change the generative power ([FS02], [WZ]).

Trivially, $\mathbb{Z}^0\mathrm{VL} = \mathbb{Q}^0\mathrm{VL} = \mathrm{CFL}$, $\mathbb{Z}^1\mathrm{VL} = \mathbb{Z}\mathrm{VL}$ and $\mathbb{Q}^1\mathrm{VL} = \mathbb{Q}\mathrm{VL}$. Furthermore, we have for each $i \in \mathbb{N}$ ([Pau80], [FS02]):

$$\mathbb{Z}^i\mathrm{VL} \subsetneq \mathbb{Z}^{i+1}\mathrm{VL} \subsetneq \mathbb{Q}\mathrm{VL} = \mathbb{Q}^i\mathrm{VL} = \mathrm{UVL}.$$

Since the proofs for these inclusions resp. equalities are constructive, we can conclude from the results on UVL that these classes are parsable in polynomial time, they contain only semilinear languages, and they are not mildly context-sensitive.

1.4.3. Coupled Nonterminal Symbols

The third type of classes we are looking at can be viewed as a restricted version of scattered context grammars. In SCG, we force the rules to be applied in predefined groups. Now, we additionally group the nonterminals in the sentential forms, based on the derivation step that introduced them, and require the rule groups to be applied to a grouped set of nonterminals instead of an arbitrary set of instances.

In this view, *multiple context-free grammars* (MCFG; [KSF88]) are unordered scattered context grammars where:

- The set of nonterminals is partitioned into groups, with the start symbol forming a group of its own.
- For each matrix, the left-hand sides consist of exactly the symbols from one group.

- Each nonterminal may appear at most once on the right-hand sides of all rules in a matrix combined.
- Terminal symbols may appear on the right-hand sides without restriction.

Rule matrices are then applied to sets of nonterminals that were introduced in the same step. If some of the nonterminals in the group to be replaced were not introduced in the respective step, the corresponding rules are skipped as in appearance checking mode.

A *coupled context-free grammar* (CCFG; [Gua92], [GHR92], [HP96], [Pit93]) then is a MCFG, with the following additional constraints:

- If one symbol of a group appears on the combined right-hand sides of a matrix, then the complete group appears.
- The nonterminal symbols in each group are ordered, and if the right-hand sides of a matrix are concatenated in the order implied by the left-hand sides, all appearing groups of nonterminals appear in order.
- If one symbol from group A appears between symbols B_i and B_{i+1} from group B in the concatenated right-hand sides, then all symbols from group A appear between B_i and B_{i+1}.

In [SMFK91], Seki et. al. show that each MCFG G can be transformed into an equivalent MCFG G' in a normal form that satisfies the first additional constraint for CCFG and has no ϵ-rules. If G is a CCFG then G' is a CCFG as well.

From this normal form construction, it is immediately clear that allowing or forbidding ϵ-rules does not change the generative power of these classes. Additionally, the normal form guarantees that we can always apply either all of the rules of a given matrix or none of them. Thus, appearance checking also has no effect on the generative power.

We get for the hierarchy that ([Gua92], [Mic05], [KSF88])

$$\text{CFL} \subsetneq \text{CCFL} \subsetneq \text{MCFL} \subsetneq \text{CSL}.$$

Finally, let us note that both classes satisfy the conditions for mildly context-sensitive languages ([Kal10]).

1.4.4. Overview

Figure 1.1 gives an overview of the language classes introduced in this section. For all classes depicted above pRCL, the membership problem is NP-hard, for those below pRCL, it is solvable in polynomial time. As mentioned before, the question appears to be open for pRCL.

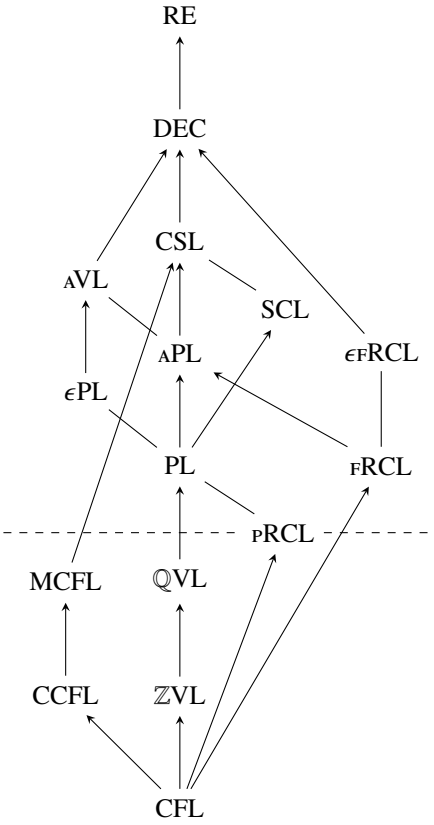
More detailed information on the classes presented in Sections 1.4.1 and 1.4.2 can be found in [DP89] and [DPS97]. For the classes from Section 1.4.3 and related formalisms, [Kal10] provides a good overview.

1.5. New Idea: Position and Length Restrictions

Inspired by the application presented in Chapter 8, we came up with a new type of restriction. We allow or disallow a rule application based on the length of the subword that eventually results from the rule application and/or the position in the complete word that this subword has.

The concept is best explained with parse trees. Figure 1.2 shows an abstract parse tree with one rule application ($A \to \alpha_1 \cdots \alpha_k$) made explicit. The labels on the leaves below this rule application are what we will call the subword generated by the rule application or the yield of the rule application. The *length of the rule application*, indicated by l in the picture, is then the length of this subword.

Regarding the position of the rule application, we take two metrics. The *f-distance* (as a shorthand for "distance to the front") is the number of characters left of the yield in the final word, the *e-distance* (short for "distance to the end") is correspondingly characterized by the number of characters to the right of the yield.



$$\begin{aligned}
\text{RE} &= \epsilon\text{CL} = \epsilon\text{SCL} = \epsilon\text{RCL} = \epsilon\text{APL} \\
&= \epsilon\text{AML} = \epsilon\text{AUML} = \epsilon\text{SCL} = \epsilon\text{ASCL} \\
&= \epsilon\text{AUSCL} = \epsilon\text{AVL} = \epsilon\text{RCL} = \epsilon\text{APNL}.
\end{aligned}$$

$$\text{CSL} = \text{CL} = \text{SCL} = \text{ASCL}.$$

$$\text{AVL} = \text{APNL}.$$

$$\begin{aligned}
\text{APL} &= \text{RCL} = \text{AML} = \text{AUML} = \text{AUSCL} \\
&= \text{ARCL}.
\end{aligned}$$

$$\begin{aligned}
\epsilon\text{PL} &= \epsilon\text{ML} = \epsilon\text{UML} = \epsilon\text{USCL} = \text{VL} \\
&= \epsilon\text{VL} = \epsilon\text{RCL} = \text{PNL} = \epsilon\text{PNL}.
\end{aligned}$$

$$\text{PL} = \text{ML} = \text{UML} = \text{SCL} = \text{RCL}.$$

$$\text{PRCL} = \epsilon\text{PRCL}.$$

$$\begin{aligned}
\mathbb{Q}\text{VL} &= \mathbb{Q}^i\text{VL} = \text{UVL} = \epsilon\text{UVL} = \text{AUVL} \\
&= \epsilon\text{AUVL}.
\end{aligned}$$

$$\mathbb{Z}^i\text{VL} \subsetneq \mathbb{Z}^{i+1}\text{VL} \subsetneq \mathbb{Q}\text{VL}.$$

Figure 1.1.: Hierarchy of the language classes introduced in this section. An edge (arrow) indicates that the lower class is a (proper) subclass of the upper class. No edge or path between two classes does not necessarily imply incomparability. The dashed line indicates the boundary between classes where parsing is NP-hard (above) resp. polynomial (below).

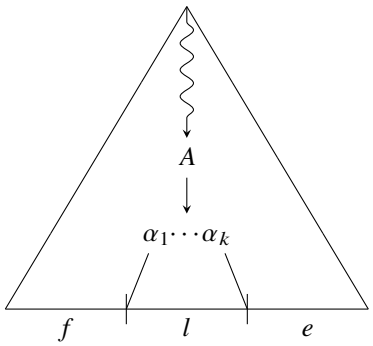


Figure 1.2.: Rule application in a context-free grammar.

Now, each rule is associated with a set of triples $M \subseteq \mathbb{N}^3$ and a rule application is only allowed, if $(f, l, e) \in M$.

Obviously, if each rule is associated with $\mathbb{N}^3$, we get a grammar that is fully equivalent to the original CFG. Thus the grammars from our new class can generate all context-free languages.

To see that they can also generate non-context-free languages, consider a grammar where the start symbol is only replaced by the rule $r = (S \to A;\ \mathbb{N} \times \mathbb{N}^{\{2\}} \times \mathbb{N})$ and where additional (unrestricted) rules generate a^* from A.

Since in this grammar, each derivation has to start with r, and r is only allowed to generate words whose length is a square number, we find that the grammar generates exactly those words from a^* whose length is a square number. That is the set $a^{\{n^2\}}$, which is known not to be context-free.

As you can see, this example only restricts the length for applying the restricted rule. The same language can be generated using restrictions on the f-distance if we replace r by $(S \to AE;\ \mathbb{N}^3)$ and $(E \to \epsilon;\ \mathbb{N}^{\{2\}} \times \mathbb{N}^2)$. This indicates that it might be interesting to examine the subclasses that result from allowing only one or two types of restrictions.

We will do this and we will also examine the subclasses that result from enforcing the different restrictions on a single rule to be independent. That means in such a grammar we do not, e.g., allow a restriction set like $\{(n, m, n) \mid n, m \in \mathbb{N}\}$ which would enforce that a certain rule can only be applied if the f- and e-distance are equal.

We will denote the resulting classes by [F][L][E][D]CFG, where the prefixes F, L and E indicate that the corresponding restriction is allowed and the prefix D indicates that we may use mutually dependent restrictions.

In addition we will consider the classes [F][L][E][D]RLING and [F][L][E][D]LLING that result from using right or left linear base grammars.

We will now proceed formally.

2. Definitions and Basic Results

2.1. Grammars

We start our formal treatment with the definition of FLEDCFG.

Definition 2.1. *A* position-and-length-dependent context-free grammar (FLEDCFG) *is a 4-tuple* $G = (N, \Sigma, R, S)$*, where*

- N *is a finite nonempty set called the* nontermial alphabet*;*
- Σ *is a finite nonempty set called the* terminal alphabet*;*
- $N \cap \Sigma = \varnothing$*;*
- $S \in N$ *is called the* start symbol*;*
- R *is a finite set of* rules *(or* productions*) of the form* $(A \to \beta;\ M)$*, where* $A \in N,\ \beta \in (N \cup \Sigma)^*$*, and* $M \subseteq \mathbb{N}^3$.

We will call the sets M *plsets* as a shorthand for position-and-length-sets. They contain the triples (f-distance, length, e-distance) for which the respective rule is *allowed* to be applied.

Note that we do not restrict the plsets in any way, even allowing the usage of sets that are not recursively enumerable. A more practically inclined reader may want to introduce a restriction to decidable or at least recursive enumerable sets to avoid questions, e.g., about how to represent the grammar if some of the plsets have no finite representation. With the exception of the parsing algorithms in Chapter 4, all of the results in this thesis hold for all three cases.

We also allow multiple rules that only differ in their plsets. Since we could replace each rule $(A \to \alpha;\ M_i)$ by the pair $(A \to A_i;\ M_i)$ and $(A_i \to \alpha;\ \mathbb{N}^3)$ with a distinct symbol A_i for each original rule, this does not influence the generative power of the grammars.[1]

To formalize the semantics of the plsets, we adapt the definition of a derivation:

For traditional CFG, one typically defines a single derivation step $\alpha \Rightarrow \beta$, where β results from α by replacing a single nonterminal according to a grammar rule. The relation $\Rightarrow^*$ is then simply the reflexive and transitive closure of $\Rightarrow$.

In the case of FLEDCFG, this approach can not be copied straightforwardly since the applicability of the rule in question typically depends on the terminal string that is eventually derived from β.

One way to circumvent this would be to start by defining complete derivations. This way, we can ensure that every rule application happens at an allowed position and length. As a downside of this approach, partial derivations can only be defined as parts of complete derivations, while for some of the proofs in the following sections, it is convenient to also include partial derivations that can not be completed.

We chose a different approach: We annotate (partial) derivations by a position/length-triple, writing $\alpha \,{}^{[(f,l,e)}\!\Rightarrow^* \beta$ to indicate that there is a way to assign yields to the nonterminal symbols of β such that when each of the symbols is derived to the assigned yield[2], the word resulting from β has a total length of l, and if the derivation is additionally embedded into a context of lengths f to the left and e to the right, all rule applications in the derivation happen at an allowed position and length.

[1] The simpler idea of replacing them with a single rule that has as plset the union of all the original plsets does no work with all subclasses (cp. Lemma 2.8).

[2] We do not care at this point whether or not such a derivation is actually possible in the grammar at hand.

Be aware that the triples annotated to the derivation are usually not equal to those corresponding to the individual rule applications. Since the latter is usually different for different steps of the derivation, such an equality cannot generally be obtained. (If α consists of a single nonterminal, the annotated triple coincides with that of the first rule application.)

As implied above, our definition will require that each step of the partial derivation assumes the same yield for the nonterminals in β when determining its validity. This restriction is easier to formally define on parse trees instead of derivations. Thus, we will start by defining parse trees and derive the definition of a derivation from that of a parse tree.

Definition 2.2. *Let $G = (N, \Sigma, R, S)$ an* FLRDCFG*, $\alpha \in N \cup \Sigma_\epsilon$, $\beta \in (N \cup \Sigma)^*$, $(f, l, e) \in \mathbb{N}^3$. A* partial parse tree from α to β at (f, l, e) in G *is a labeled, ordered tree T satisfying:*

- *The leaves of T are labeled with symbols from $N \cup \Sigma_\epsilon$.*
- *The internal nodes of T are labeled with symbols from N.*
- *The root of T is labeled with α.*
- *Concatenation of the leaves' labels in the order implied by the tree yields β.*
- *There is a function* yield *from the leaves $x_1, \ldots, x_n$ of T to Σ^* satisfying:*
  - $\text{yield}(x_i) = a$*, whenever x_i is labeled with $a \in \Sigma_\epsilon$.*
  - *If a subtree of T contains the leaves $x_i, \ldots, x_j$, the root of the subtree is labeled with A, and the children of the root are labeled with $\gamma_1, \ldots, \gamma_k$ in this order, then*
    $\exists M \subseteq \mathbb{N}^3 \colon\ (A \to \gamma_1 \cdots \gamma_k;\ M) \in R$ *and* $(f_i, l_{i,j}, e_j) \in M$*, where*
    $f_i = f + \sum_{h=1}^{i-1} |\,\text{yield}(x_h)|$, $l_{i,j} = \sum_{h=i}^{j} |\,\text{yield}(x_h)|$*, and* $e_j = e + \sum_{h=j+1}^{n} |\,\text{yield}(x_h)|$.

For $\alpha \in (N \cup \Sigma)^*$, $|\alpha| = k$, there is a partial derivation $\alpha \left\Rightarrow\!\!\!\!\!\left[\begin{smallmatrix} r_1,\ldots,r_n \\ (f,l,e) \end{smallmatrix}\right]_G \beta$ in G*, iff there are $l_1, \ldots, l_k \in \mathbb{N}$ and $\beta^{(1)}, \ldots, \beta^{(k)} \in (N \cup \Sigma)^*$ satisfying:*

- $l = \sum_{i=1}^{k} l_i$,
- $\beta = \beta^{(1)} \cdots \beta^{(k)}$,
- *for $1 \le i \le k$, there is a partial parse tree from α_i to $\beta^{(i)}$ at (f_i, l_i, e_i) in G, where $f_i = f + \sum_{j=1}^{i-1} l_j$ and $e_i = e + \sum_{j=i+1}^{k} l_j$, and*
- *those parse trees together have n internal nodes that can be enumerated $x_1, \ldots, x_n$ such that:*
  - *when x_i is the parent of x_j, then $i < j$, and*
  - *when $r_i = (A \to \gamma)$, then x_i is labeled with A and the children of x_i are labeled with $\gamma_1, \ldots, \gamma_{|\gamma|}$ in this order.*

A partial derivation is a leftmost partial derivation $\alpha \left\Rightarrow\!\!\!\!\!\left[\begin{smallmatrix} r_1,\ldots,r_n \\ (f,l,e);L \end{smallmatrix}\right]_G \beta$ *iff the enumeration from the last bullet point corresponds to the order obtained by successively traversing the parse trees for $\alpha_1, \ldots, \alpha_k$ in preorder.*

For $w \in \Sigma^*$, a full parse tree for w in G *is a partial parse tree from S to w at $(0, |w|, 0)$ in G.*

From this definition, it can easily be seen that the parse trees for an FLEDCFG G are a subset of the parse trees for the CFG G' that results from ignoring all the plsets.

Furthermore, if a given parse tree is valid for G, all derivations corresponding to this parse tree in G' are also valid derivations in G. Thus, the observation that there are usually multiple derivations (including exactly one leftmost derivation) that correspond to the same parse tree carries over from CFG to FLEDCFG.

In the following, we will switch freely between the representations as derivations and parse trees, and we will consider derivations that correspond to the same parse tree equal unless noted otherwise.

Now, let us define some convenient notations for derivations.

Definition 2.3.

- *We use* $\alpha \mathrel{=}[(f,l,e)]\Rightarrow^{n}_{G} \beta$ *as a shorthand for* $\exists r_1,\dots,r_n\colon\ \alpha \mathrel{=}\left[\begin{smallmatrix} r_1,\dots,r_n \\ (f,l,e)\end{smallmatrix}\right]\Rightarrow_{G} \beta$.
- $\alpha \mathrel{=}[(f,l,e)]\Rightarrow^{*}_{G} \beta$ *means* $\exists n\colon\ \alpha \mathrel{=}[(f,l,e)]\Rightarrow^{n}_{G} \beta$.
- $\alpha \mathrel{=}[(f,l,e)]\Rightarrow_{G} \beta$ *means* $\alpha \mathrel{=}[(f,l,e)]\Rightarrow^{1}_{G} \beta$.
- *If* $\beta \in \Sigma^*$ *and* $f = e = 0$*, we write* $\alpha \Rightarrow^{\cdot}_{G} \beta$ *instead of* $\alpha \mathrel{=}[(0,|\beta|,0)]\Rightarrow^{\cdot}_{G} \beta$*, for* $\cdot$ *any of the superscripts used above.*
- *If we write a (partial) derivation as multiple subparts we may drop the (identical) position/length-indicator from all but one of the parts as in* $\alpha_1 \Rightarrow^{n}_{G} \alpha_2 \mathrel{=}[(f,l,r)]\Rightarrow_{G} \alpha_3$.
- *If it is obvious from the context which grammar is meant, we drop the index G.*

Now, we are ready to define the language generated by an fledCFG.

Definition 2.4. *The* language generated by G *is the set* $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^{*}_{G} w\}$.

To get a better grasp of the possibilities of fledCFG, lets look at some examples.

Example 2.5. *Let* $G_1 = (\{A,B,C\},\{a,b,c\},R,A)$*, with*

$$\begin{aligned} R = \{&(A \to aA;\ \mathbb{N}^3),\ (A \to B;\ \{(n,2n,m) \mid n,m \in \mathbb{N}\}),\\ &(B \to bB;\ \mathbb{N}^3),\ (B \to C;\ \{(2n,n,m) \mid n,m \in \mathbb{N}\}),\\ &(C \to cC;\ \mathbb{N}^3),\ (C \to \epsilon;\ \mathbb{N}^3)\}. \end{aligned}$$

In order to intuitively determine the language generated by an fledCFG, it is often convenient to first determine the superset of the language generated by ignoring the restrictions. In the case of G_1*, we find that the grammar generates words of the form* $a^{n_1}b^{n_2}c^{n_3}$.

Looking at the restrictions, we then note that the rule $(B \to C)$ *is applied only once and the subword generated from this application is* c^{n_3}*, having a length of* n_3*. Also, the* f*-distance of the application is* $n_1 + n_2$ *and the* e*-distance is* 0*. Thus, this rule application (and therefore the whole derivation) is only allowed if* $n_1 + n_2 = 2n_3$.

Similarly, $(A \to B)$ *yields* $b^{n_2}c^{n_3}$*. This yield has a length of* $n_2 + n_3$*,* f*-distance of* n_1*, and* e*-distance of* 0*. Hence, the plset of the rule translates into the requirement that* $2n_1 = n_2 + n_3$.

Combining both equations we find $n_1 = n_2 = n_3$*, and thus* $L(G_1) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$.

Example 2.6. *For a slightly more complex example, let*

$$G_2 = (\{S\},\ \{a\},\ \{(S \to SS;\ \{\mathbb{N} \times \{2\}^{\mathbb{N}} \times \mathbb{N}\}), (S \to a;\ \mathbb{N}^3)\},\ S).$$

Here, ignoring the restrictions, we find that this grammar can produce strings of the form a^+.

We then note that every derivation except $S \Rightarrow a$ *starts with the first rule. This implies that* $L(G_2)$ *can only contain words with a length of* 2^l*,* $l \in \mathbb{N}$*. To see that all of those are actually in* $L(G_2)$*, consider the following construction of* $w = a^{2^l}$ *in* $l+1$ *phases:*

In phase $i \le l$*, replace each* S *introduced in phase* $i-1$ *by* SS*. Finally in phase* $l+1$ *replace each* S *by* a*. This way, in phase* i*, each rule is applied with length* 2^{l+1-i}*, and the application is thus allowed.*

Looking even more closely, we note that the length of the yield of each nonterminal appearing during a derivation in G_2 *has to be a power of 2. If the nonterminal is replaced according to the first rule, this is enforced by the plset, while for the second rule, this length is fixed to* 2^0 *by the structure of the rule.*

However, since nonterminals are always introduced in pairs by the first rule and the sum of two powers of 2 is itself a power of 2 if and only if the powers summed up are equal, the derivations described by the construction above are actually the only valid derivations in G_2 *(except for changing the order of rule applications).*

It can be seen in the examples that we can generate non-context-free languages without using all possibilities of plsets. Some of the rules are completely unrestricted (those with plsets $\mathbb{N}^3$). The rule $(S \to SS;\ \mathbb{N}\times\{2\}^{\mathbb{N}}\times\mathbb{N})$ in G_2 may be applied at arbitrary positions, but only to generate words of certain lengths. The rules $(A \to B)$ and $(B \to C)$ in G_1, on the other hand not only depend on two criteria, they also require them to satisfy a certain relation.

This leads to the question if an increased complexity of the plsets actually allows more languages to be described. For example, $L(G_2)$ can also be generated by the grammars
$G_2' = (\{S\},\ \{a\},\ \{(S \to aS;\ \mathbb{N}^3), (S \to \epsilon;\ \{2\}^{\mathbb{N}} \times \mathbb{N}^2)\},\ S)$ and
$G_2'' = (\{S\},\ \{a\},\ \{(S \to Sa;\ \mathbb{N}^3), (S \to \epsilon;\ \mathbb{N}^2 \times \{2\}^{\mathbb{N}})\},\ S)$.
In order to be able to study this question, we define subclasses of the class FLEDCFG. As mentioned in the introduction, the names for theses subclasses will be derived from FLEDCFG in a straightforward way:

- The F is dropped for classes which allow no restrictions of the f-distance.
- The L is dropped for classes which allow no length restrictions.
- The E is dropped for classes which allow no restrictions of the e-distance.
- The D is dropped for classes which only allow restrictions for f-distance, length and e-distance that are independent of each other. This is trivially the case if only one (or no) type of restriction is allowed.

Formally, these are constraints on the plsets:

Definition 2.7. *An* FLEDCFG $G = (N, \Sigma, R, S)$ *is called an*

- LEDCFG *if* $\forall (A \to \alpha;\ M) \in R\colon \exists M_{l,e} \subseteq \mathbb{N}^2\colon\ M = \mathbb{N} \times M_{l,e}$,
- FEDCFG *if* $\forall (A \to \alpha;\ M) \in R\colon \exists M_{f,e} \subseteq \mathbb{N}^2\colon\ M = \{(f, l, e) \mid (f, e) \in M_{f,e},\ l \in \mathbb{N}\}$,
- FLDCFG *if* $\forall (A \to \alpha;\ M) \in R\colon \exists M_{f,l} \subseteq \mathbb{N}^2\colon\ M = M_{f,l} \times \mathbb{N}$,
- FLECFG *if* $\forall (A \to \alpha;\ M) \in R\colon \exists M_f, M_l, M_e \subseteq \mathbb{N}\colon\ M = M_f \times M_l \times M_e$,
- LECFG *if it is an* LEDCFG *and an* FLECFG,
- FECFG *if it is an* FEDCFG *and an* FLECFG,
- FLCFG *if it is an* FLDCFG *and an* FLECFG,
- F[D]CFG *if it is an* FLDCFG *and an* FEDCFG,
- L[D]CFG *if it is an* FLDCFG *and an* LEDCFG *and*
- E[D]CFG *if it is an* FEDCFG *and an* LEDCFG.

We use the names without D in the prefix for the classes with only one type of restriction, but the other may appear when using the notation with prefixes in brackets. Analogously, we will treat DCFG as synonymous to CFG, and treat a grammar where each plset is $\mathbb{N}^3$ as equal to the same grammar without plsets.

In some of the proofs later on, we will construct plsets for new grammars as intersections of existing ones. The following lemma establishes that the resulting grammars will be in the same subclasses as the original ones.

Lemma 2.8. *If* $M_1 \subseteq \mathbb{N}^3$ *and* $M_2 \subseteq \mathbb{N}^3$ *satisfy one or more of the constraints from Definition 2.7, then* $M_1 \cap M_2$ *will also satisfy these constraints. With the exception of the constraint for* FLECFG *(and those that inherit it), this also holds for* $M_1 \cup M_2$.

Proof. The closures are immediate from Definition 2.7. As an example for the nonclosure of FLECFG, note that $\{1\}^3 \cup \{2\}^3 = \{(1, 1, 1), (2, 2, 2)\}$ does not satisfy the constraint for FLE-plsets. □

We can (and will) work around the nonclosure of FLE-plsets under union by exploiting the fact that Definition 2.1 allows for multiple rules that only differ in their plsets.

Since right and left linear grammars are special cases of context-free grammars, restricted right and left linear grammars arise naturally as special cases of restricted context-free grammars.

Definition 2.9. *An* [F][L][E][D]CFG $G = (N, \Sigma, R, S)$ *is*

- right linear*(an* [F][L][E][D]RLING*) if each rule is of the form* $(A \to uB)$ *or* $(A \to u)$*, where* $A, B \in N$ *and* $u \in \Sigma^*$*,*
- left linear *(an* [F][L][E][D]LLING*) if each rule is of the form* $(A \to Bu)$ *or* $(A \to u)$*, where* $A, B \in N$ *and* $u \in \Sigma^*$*.*

Example 2.10. G_1 *from Example 2.5 is an* FLDRLING *but not an* FLERLING*.*

Due to the special structure of left linear grammars, the application of any production will always happen at f-distance 0. Thus, by replacing each rule $(A \to \alpha;\ M)$ by $(A \to \alpha;\ \{(f, l, e) \mid (0, l, e) \in M,\ f \in \mathbb{N}\})$, we can always get an equivalent grammar that does not have f-distance-restrictions. Analogously, the application of a rule in right linear grammars always happens at e-distance 0, allowing those restrictions to be eliminated. This immediately yields the following equalities.

Lemma 2.11.

- F[L][E][D]LLINL=[L][E][D]LLINL*,*
- [F][L]E[D]RLINL=[F][L][D]RLINL*.*

It is a well-known observation that reversing each production of a context-free grammar yields a context-free grammar that generates the reversal of the language generated by the original grammar. Furthermore, it follows immediately from Definition 2.2 that on reversing f- and e-distances exchange their role, while length-dependencies are unaffected. Thus we find:

Lemma 2.12.

- *The classes* F[L]E[D]CFL *and* LCFL *are closed under reversal.*
- $L \in$ F[L][D]CFL $\hookleftarrow$ $L^R \in$ [L]E[D]CFL*,*
- $L \in$ LRLINL $\hookleftarrow$ $L^R \in$ LLLINL*,*
- $L \in$ F[L][D]RLINL $\hookleftarrow$ $L^R \in$ [L]E[D]LLINL*.*

For some of the proofs in the following sections, it will be convenient if our grammars are in a normal form. Thus, we define:

Definition 2.13. *An* FLEDCFG $G = (N, \Sigma, R, S)$ *is called*

- ϵ-free*, iff* R *contains no rules of the form* $(A \to \epsilon;\ M)$ *except possibly* $(S \to \epsilon;\ M)$*, and* S *does not appear on the right-hand side of any rule.*
- chain-free*, iff there is no rule of the form* $(A \to B;\ M)$*,* $A, B \in N$*.*
- *in* Chomsky Normal Form (CNF)*, iff each rule is of one of the three forms* $(A \to a;\ M)$*,* $A \in N$*,* $a \in \Sigma$*,* $(A \to BC;\ M)$*,* $A \in N$*,* $B, C \in N \setminus \{S\}$*, or* $(S \to \epsilon;\ M)$*.*

It is well-known that each CFG can be transformed into an equivalent grammar in CNF. We will now adapt this transformation to FLEDCFG.

Lemma 2.14. *For any* [F][L][E][D]CFG G*, there is an* ϵ*-free* [F][L][E][D]CFG G' *with* $L(G') = L(G)$.

Proof. Let $G = (N, \Sigma, R, S)$ and $E = \{A \in N \mid \exists f, e\colon\ A \Rightarrow\!\!\!\!\!\!\lceil (f, 0, e) \rfloor\!\!\!\!\!\!\Rightarrow^* \epsilon\}$.

The basic idea for the construction of G' is the same as for traditional CFG (cf. [Har78, Theorem 4.3.1]): All ϵ-rules are removed, and instead, new rules are added that result from original rules by leaving out any combination of symbols that can be derived to ϵ. Intuitively the new rules combine the partial derivation $A \Rightarrow^* \epsilon$ with the rule that originally introduced A.

The main problem when translating this construction to FLEDCFG is the following: If a symbol that can be derived to ϵ is removed, we usually will not have immediate access to the position it appeared at. As a simple example, take the grammar with the rules $(S \to ACB;\ \mathbb{N}^3)$, $(C \to \epsilon;\ \{(n, 0, n) \mid n \in \mathbb{N}\})$ and additional rules that derive A to a^* and B to b^*. This grammar generates the language $a^n b^n$. Applying the conversion described above will remove the second rule and instead add a rule $(S \to AB;\ \mathbb{N}^3)$. But it is not possible to add a plset to this rule that ensures that the number of a's and b's are the same.

A first idea to counter this problem is to replace A in the modified rule by a symbol A_C, where the index C indicates that a partial derivation $C \Rightarrow^* \epsilon$ was removed to the right of A, and for each rule $(A \to \alpha;\ M)$ add a rule $(A_C \to \alpha;\ M')$, where $M' = M \cap \{(f, l, e) \mid (f + l, 0, e) \in M_C\}$, and M_C is a plset describing where the partial derivation $C \Rightarrow^* \epsilon$ is allowed.

This idea works for the most part, but there are some aspects we need to deal with:

In order to see how to determine the sets M_C, note that all rules in a given subtree that yields ϵ are applied with the same f-distance f, length 0, and e-distance $e = |w| - f$. Thus, the derivation is allowed for exactly those triples $(f, 0, e)$ that lie in the intersection of the plsets of all rules appearing in the partial derivation.

If there are multiple partial derivations $C \Rightarrow^* \epsilon$, the rules $(A_C \to \alpha)$ should be conditioned to at least one of the partial derivations being allowed at the position in question. This can be ensured by defining M_C as the union of all the intersected plsets. However, as we have seen in Lemma 2.8, not all subclasses of plsets are closed under union. Thus, we define M_C as the set of all the intersections and introduce a separate rule $(A_C \to \alpha)$ for each entry in M_C.

Formally, let

$$M_C = \left\{ \bigcap_{1 \le i \le k} M_i \;\middle|\; \exists f, e\colon\ C \Rightarrow\!\!\!\!\!\!\left\lceil \substack{(A_1 \to \alpha_1;\ M_1), \ldots, (A_k \to \alpha_k;\ M_k) \\ (f,0,e)} \right\rfloor\!\!\!\!\!\!\Rightarrow \epsilon \right\}$$

for each $C \in N$. Note that M_C is guaranteed to be finite since G has $|R|$ different plsets which can be combined to at most $2^{|R|}$ different intersections. For the classes [F][L][E]DCFG where the plsets are closed under union, we can simplify the construction by using

$$M'_C = \bigcup_{M \in M_C} M.$$

The next case we have to consider concerns multiple ϵ-derivations neighboring each other. Extending the example above, consider the rules $(S \to ACDB)$, $(C \to \epsilon)$, and $(D \to \epsilon)$. Here, we introduce rules $(S \to A_{\{C,D\}}B;\ M')$, as well as rules $(S \to A_C DB)$ and $(S \to AC_D B)$ to care for the derivations where C resp. D are derived to something other than ϵ. In the simpler case, we introduce only one rule, setting $M' = M \cap \{(f, l, e) \mid (f + l, 0, e) \in M'_C \cap M'_D\}$. For the construction using M_C and M_D, we have to add a separate instance of the new rule for each combination of one plset from M_C and one from M_D.

Note that M' does not change if the order of the removed symbols changes, or if multiple instances of the same symbol are removed. This justifies the use of a set of nonterminals instead of a string as index. The generalization to more than two symbols should be obvious by now. Refer to the formal definition of G' below for the technical details.

Another special case arises if the symbol to be removed is the leftmost symbol of the rule. If it is the only symbol of the rule, the resulting rule is itself an ϵ-rule that will be discarded. Otherwise, we can move the check to the leftmost of the remaining symbols, symmetric to the construction above. So for rules $(S \to CA)$ and $(C \to \epsilon)$, we add $(S \to A'_C)$, and for each rule $(A \to \alpha;\ M)$ we add $(A'_C \to \alpha;\ M')$, where $M' = M \cap \{(f, l, e) \mid (f, 0, l + e) \in M'_C\}$.

As you can see from the rules $(S \to CAC)$ and $(C \to \epsilon)$, it may be necessary to perform both checks on a single symbol. In this case, the resulting plset will be the intersection of the original plset and both plsets for additional checks.

In order to avoid having to introduce several classes of nonterminals depending on which types of checks have to be performed for them, our formal definition will only contain nonterminals of the form $\alpha_{\mathcal{L},\mathcal{R}}$, $\mathcal{L}, \mathcal{R} \subseteq N$, where $\alpha \in (N \cup \Sigma)$. Here, $\mathcal{L}$ denotes the nonterminals derived to ϵ directly to the left of α, if α is the leftmost remaining symbol and $\mathcal{R}$ denotes the nonterminals derived to ϵ directly to the right of α. If there are no such symbols, resp. α is not leftmost in the rule, the respective index will be $\varnothing$ and not influence the resulting plsets. For symbols $\alpha_{\mathcal{L},\mathcal{R}},\ \alpha \in \Sigma$, we add rules $(\alpha_{\mathcal{L},\mathcal{R}} \to \alpha)$ that perform exactly the checks implied by $\mathcal{L}$ and $\mathcal{R}$.

For FLEDCFG, this is already sufficient. But for the subclasses, there is one more pitfall to circumvent: Assume we are given an FCFG with a rule $(A \to \alpha;\ \mathbb{N}^3)$ and $M'_C = \mathbb{N}^{\{2\}} \times \mathbb{N}^2$. Furthermore assume that C appears to the right of A in some other rule of the grammar. Then, our construction introduces the rule $(A_{\varnothing,\{C\}} \to \alpha;\ M')$, $M' = \{(f, l, e) \in \mathbb{N}^3 \mid f + l \in \mathbb{N}^{\{2\}}\}$. But M' is not an F-plset or even an FLE-plset as it contains $(1, 3, 1)$ and $(3, 1, 1)$, but not $(1, 1, 1)$.

This problem would be avoided if the checks were performed on rules that yield a subword of a fixed length immediately to the left of the removed subderivation. In this case, the f-distance from the ϵ-derivation becomes an f-distance for the new rule by simply subtracting the fixed length, the e-distance can be carried over as is, and we do not need any length restrictions.

Now, observe that the grammars resulting from our construction only introduce terminal symbols in rules $(a_{\mathcal{L},\mathcal{R}} \to a)$. Thus, the rule introducing the terminal directly to the left of the removed ϵ-derivation has the desired properties. All we have to do is ensure that the check is performed on this rule, which is achieved by adding the removed symbol to its set $\mathcal{R}$. When removing C from a rule $(B \to \alpha_1 AC\alpha_2)$, the symbol in question will be the rightmost leaf in the partial parse tree with root A. So, we add the replaced symbol to the set $\mathcal{R}$ of A as before. If A already is a terminal symbol, we are done. Otherwise, instead of replacing rules $(A \to \alpha;\ M)$ by rules $(A_{\mathcal{L},\mathcal{R}} \to \alpha';\ M')$, where $\alpha' = (\alpha_1)_{\mathcal{L}_1,\mathcal{R}_1} \cdots (\alpha_k)_{\mathcal{L}_k,\mathcal{R}_k}$, we replace them by $(A_{\mathcal{L},\mathcal{R}} \to \alpha'';\ M)$, where α'' results from α' by adding the elements of $\mathcal{R}$ to $\mathcal{R}_k$. This way, the information that C has been removed will be handed down the right flank of the partial parse tree until it reaches the rightmost leaf as desired.

If the removed symbol is the leftmost symbol of its rule, we can move the check to the terminal-introducing rule to the right of the removed subtree. Analogous to the previous paragraph, we can get the information about the removed symbol to the intended position in the parse tree by handing down the contents of $\mathcal{L}$ from the symbol on the left-hand side to the set $\mathcal{L}_1$ in each rule.

Figure 2.1 illustrates the construction for the general case of a group of neighboring subtrees that only yield ϵ. Typically, not all of the subtrees depicted will actually be present.

Finally, we introduce a new start symbol S' that only appears in the rules $(S' \to S_{\varnothing,\varnothing};\ \mathbb{N}^3)$ and, if $S \in E$, $(S' \to \epsilon;\ M)$ for each $M \in M_S$.

Formally, $G' = (N', \Sigma, R', S')$, where $N' = \{\alpha_{L,R} \mid \alpha \in (N \cup \Sigma),\ L, R \subseteq E\} \cup \{S'\}$ and

$$
\begin{aligned}
R' = \{&(A_{\mathcal{L},\mathcal{R}} \to (\alpha_{i_1})_{\mathcal{L}',\mathcal{R}_1}(\alpha_{i_2})_{\varnothing,\mathcal{R}_2} \cdots (\alpha_{i_k})_{\varnothing,\mathcal{R}_k};\ M) \mid \\
&\quad (A \to \alpha_1 \cdots \alpha_n;\ M) \in R,\ n > 0,\ 1 \le i_1 < \cdots < i_k \le n,\ l \notin \{i_j \mid 1 \le j \le k\} \rightharpoonup \alpha_l \in E, \\
&\quad \mathcal{L}' = \mathcal{L} \cup \{\alpha_l \mid 1 \le l < i_1\},\ \mathcal{R}_j = \{\alpha_l \mid i_j < l < i_{j+1}\}, 1 \le j < k,\ \mathcal{R}_k = \mathcal{R} \cup \{\alpha_l \mid i_k < l \le n\}\} \\
\cup\ \{&(a_{\mathcal{L},\mathcal{R}} \to a;\ M_{\mathcal{L}} \cap M_{\mathcal{R}}) \mid \mathcal{L} = \{A_{\mathcal{L},i} \mid 1 \le i \le |\mathcal{L}|\} \subseteq E,\ \mathcal{R} = \{A_{\mathcal{R},i} \mid 1 \le i \le |\mathcal{R}|\} \subseteq E, \\
&\quad a \in \Sigma,\ \exists M_{\mathcal{L},1} \in M_{A_{\mathcal{L},1}}, \ldots, M_{\mathcal{L},|\mathcal{L}|} \in M_{A_{\mathcal{L},|\mathcal{L}|}}, M_{\mathcal{R},1} \in M_{A_{\mathcal{R},1}}, \ldots, M_{\mathcal{R},|\mathcal{R}|} \in M_{A_{\mathcal{R},|\mathcal{R}|}} : \\
&\quad\quad M_{\mathcal{L}} = \bigcap_{1 \le i \le |\mathcal{L}|} \{(f, l, e) \mid (f, 0, e + 1) \in M_{\mathcal{L},i}\},\ M_{\mathcal{R}} = \bigcap_{1 \le i \le |\mathcal{R}|} \{(f, l, e) \mid (f + 1, 0, e) \in M_{\mathcal{R},i}\}\} \\
\cup\ \{&(S' \to S_{\varnothing,\varnothing};\ \mathbb{N}^3)\} \cup \{(S' \to \epsilon;\ M) \mid M \in M_S\}.
\end{aligned}
$$

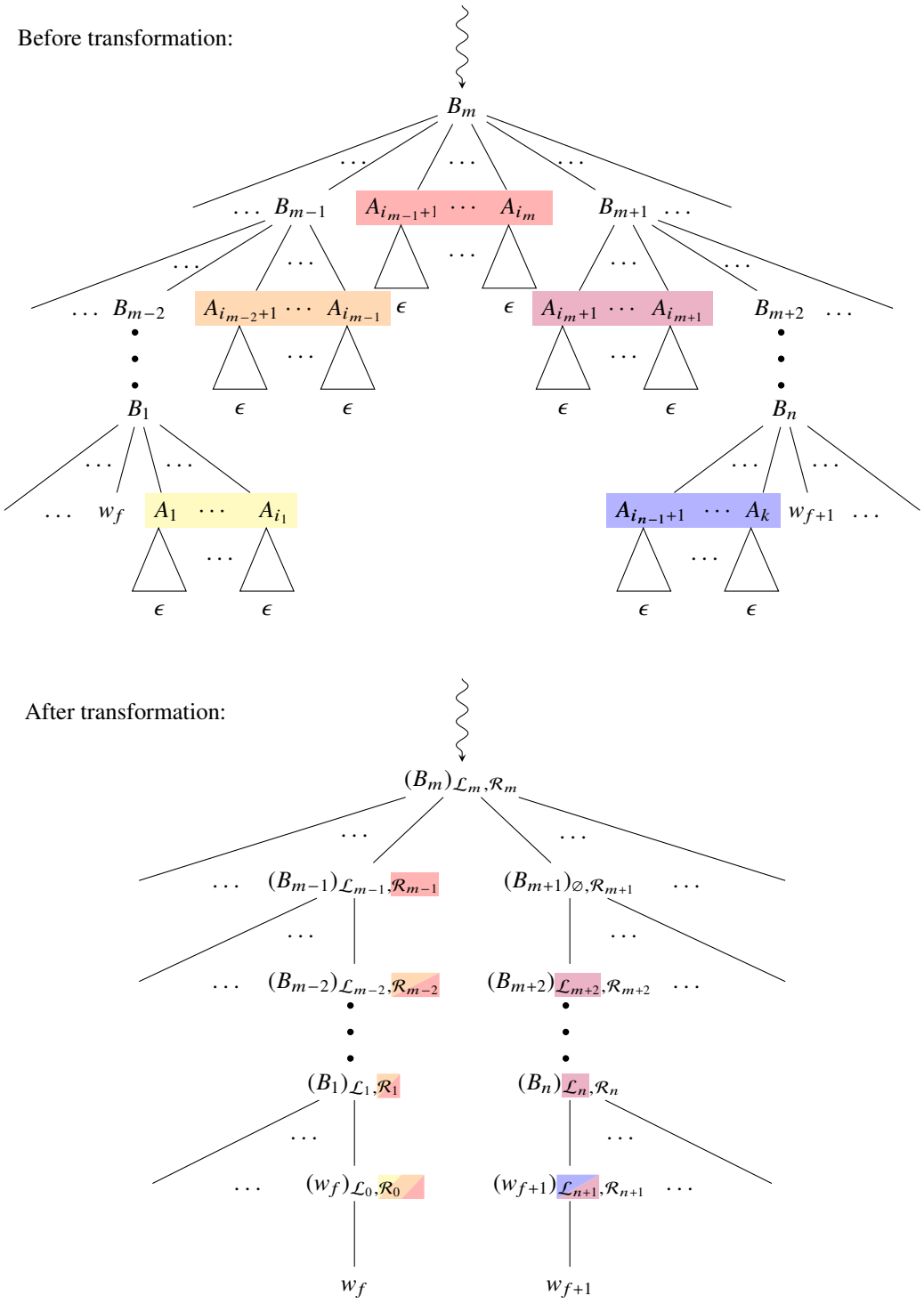


Figure 2.1.: Partial parse tree with all possible ϵ-subtrees between w_f and w_{f+1} (top), and corresponding partial parse tree for the ϵ-free grammar as constructed in Lemma 2.14 (bottom). In the second tree, $\mathcal{R}_s = \{A_j \mid i_s < j \leq i_m\}$, $0 \leq s < m$, $i_0 = 1$, and $\mathcal{L}_s = \{A_j \mid i_m < j \leq i_{s-1}\}$, $m+1 < s \leq n+1$, as indicated by the colors. The contents of the other sets $\mathcal{R}_s$, $\mathcal{L}_s$ depend on parts of the tree not pictured. Note that $\mathcal{R}_0 \cup \mathcal{L}_{n+1} = \{A_j \mid 1 \leq j \leq k\}$.

For [F][L][E]DCFG, we may instead use the slightly simpler definition based on the sets M'_C,

$$\begin{aligned}R' = \{&(A_{\mathcal{L},\mathcal{R}} \to (\alpha_{i_1})_{\mathcal{L}',\mathcal{R}_1}(\alpha_{i_2})_{\emptyset,\mathcal{R}_2}\cdots(\alpha_{i_k})_{\emptyset,\mathcal{R}_k};\ M) \mid \\ &(A \to \alpha_1\cdots\alpha_n;\ M) \in R,\ n>0,\ 1 \le i_1 < \cdots < i_k \le n,\ l \notin \{i_j \mid 1 \le j \le k\} \rightsquigarrow \alpha_l \in E,\\ &\mathcal{L}' = \mathcal{L} \cup \{\alpha_l \mid 1 \le l < i_1\},\ \mathcal{R}_j = \{\alpha_l \mid i_j < l < i_{j+1}\}, 1 \le j < k,\ \mathcal{R}_k = \mathcal{R} \cup \{\alpha_l \mid i_k < l \le n\}\}\\ \cup\ \{&(a_{\mathcal{L},\mathcal{R}} \to a;\ M_{\mathcal{L}} \cap M_{\mathcal{R}}) \mid \mathcal{L},\mathcal{R} \subseteq E,\ a \in \Sigma,\\ &M_{\mathcal{L}} = \bigcap_{C\in\mathcal{L}}\{(f,l,e) \mid (f,0,e+1) \in M'_C\},\ M_{\mathcal{R}} = \bigcap_{C\in\mathcal{R}}\{(f,l,e) \mid (f+1,0,e) \in M'_C\}\}\\ \cup\ \{&(S' \to S_{\emptyset,\emptyset};\ \mathbb{N}^3),\ (S' \to \epsilon;\ M_S)\}.\end{aligned}$$

In both cases, none of the rules in R' have S' on their right-hand side, and only the rules $(S' \to \epsilon;\ M)$ have ϵ as their right-hand side. Thus, G' is ϵ-free.

$L(G) = L(G')$ follows from the considerations above.

Now assume, M is a plset that satisfies the conditions for one of the subclasses from Definition 2.7. We show that $M_1 = \{(f,l,e) \mid (f,0,e+1) \in M\}$ and $M_2 = \{(f,l,e) \mid (f+1,0,e) \in M\}$ also satisfy those conditions. From this and Lemma 2.8, we can conclude that G' is in the same subclasses as G.

LEDCFG: Assume, $M = \mathbb{N} \times M_{l,e}$, $M_{l,e} \subseteq \mathbb{N}^2$. Then,
$M_1 = \mathbb{N}^2 \times M'_e$, where $M'_e = \{e \mid (0,e+1) \in M_{l,e}\}$, and
$M_2 = \mathbb{N}^2 \times M''_e$, where $M''_e = \{e \mid (0,e) \in M_{l,e}\}$.

FEDCFG: Assume, $M = \{(f,l,e) \mid (f,e) \in M_{f,e},\ l \in \mathbb{N}\}$, $M_{f,e} \subseteq \mathbb{N}^2$. Then,
$M_1 = \{(f,l,e) \mid (f,e+1) \in M_{f,e},\ l \in \mathbb{N}\} = \{(f,l,e) \mid (f,e) \in M_{f,e} + \{(0,1)\},\ l \in \mathbb{N}\}$, and
$M_2 = \{(f,l,e) \mid (f+1,e) \in M_{f,e},\ l \in \mathbb{N}\} = \{(f,l,e) \mid (f,e) \in M_{f,e} + \{(1,0)\},\ l \in \mathbb{N}\}$.

FLDCFG: Assume, $M = M_{f,l} \times \mathbb{N}$, $M_{f,l} \subseteq \mathbb{N}^2$. Then,
$M_1 = M'_f \times \mathbb{N}^2$, where $M'_f = \{f \mid (f,0) \in M_{f,l}\}$ and
$M_2 = M''_f \times \mathbb{N}^2$, where $M''_f = \{f \mid (f+1,0) \in M_{f,l}\}$.

FLECFG: Let $M = M_f \times M_l \times M_e$.
If $0 \in M_l$, then $M_1 = M_f \times \mathbb{N} \times (M_e - \{1\})$, and $M_2 = (M_f - \{1\}) \times \mathbb{N} \times M_e$.
If $0 \notin M_l$, then $M_1 = M_2 = \emptyset \times \emptyset \times \emptyset$. □

Lemma 2.15. *For every* [F][L][E][D]CFG G, *there is an* ϵ*-free and chain-free* [F][L][E][D]CFG G' *satisfying* $L(G') = L(G)$.

Proof. Let $G = (N, \Sigma, R, S)$.

Again, we use the same idea as for traditional CFG (cf. [Har78, Theorem 4.3.2]): If $A \Rightarrow^* B$ is possible in G, we add a rule $(A \to \alpha)$ for each rule $(B \to \alpha)$ in G, intuitively combining the chain with the following rule.

Since chain-rules are applied with the same position and length as the following rule, it suffices to use the intersections of the plsets of all rules involved as the plset of the combined rule to ensure that all checks are still performed. As in the previous lemma, we create multiple rules if there are multiple chains to guarantee G' stays in the correct subclass.

Formally, for each pair $A, B \in N$, let

$$M_{A,B} = \left\{ \bigcap_{0\le i\le k} M_i \mid \exists f,l,e\colon\ A \Rightarrow\!\!\!\!\!\!\left[\begin{matrix}(A\to A_1;\ M_0),\ldots,(A_k\to B;\ M_k)\\ (f,l,e)\end{matrix}\right]\!\!\!\Rightarrow B \right\},$$

including $\mathbb{N}^3$ in each set $M_{A,A}$. Then, $G' = (N, \Sigma, R', S)$, where

$$R' = \{(A \to \alpha;\ M_1 \cap M_2) \mid (B \to \alpha;\ M_1) \in R,\ \alpha \notin N,\ M_2 \in M_{A,B}\}.$$

Due to Lemma 2.14, we may assume that G is ϵ-free. Since the right-hand sides of rules in R' all co-occur in R, G' inherits this property. Since by construction, there are no rules with a single nonterminal as the right-hand side in R', it is also chain-free.

$L(G') = L(G)$ follows from the above considerations and with Lemma 2.8 we can conclude from the construction that G' is in the desired subclasses. □

Lemma 2.16. *For any* [F][L][E][D]CFG G*, there is an* [F][L][E][D]CFG G' *in CNF with* $L(G') = L(G)$.

Proof. Let $G = (N, \Sigma, R, S)$. By Lemma 2.15, we may assume that G is ϵ-free and chain-free.

Once again, we can adopt the well-known construction from traditional CFG (cf. [Har78, Theorem 4.4.1]). We remove terminal symbols from right-hand sides by introducing new nonterminals that can only be derived to the terminal they replace, and we split up longer right-hand sides into chains of rules that create the symbols one after the other.

Since the first rule of each chain will be applied with the same positions and length as the original rule, it can simply inherit the plset while the other rules are unrestricted.

Formally, let

$$h(\alpha) = \begin{cases} A_\alpha & \alpha \in \Sigma, \\ \alpha & \alpha \in N, \\ h(\alpha_1) \cdot h(\alpha_{2\dots|\alpha|}) & |\alpha| > 1, \end{cases}$$

and $G' = (N', \Sigma, R', S)$, where

$$N' = N \sqcup \{A_a \mid a \in \Sigma\} \sqcup \{A_{r,i} \mid r = (A \to \alpha;\ M) \in R,\ 1 \le i \le |\alpha| - 2\} \text{ and}$$

$$\begin{aligned} R' = {} & \{(A \to h(\alpha);\ M) \mid (A \to \alpha;\ M) \in R,\ |\alpha| \le 2\} \\ & \cup \{(A \to h(\alpha_1)A_{r,1};\ M) \mid r = (A \to \alpha;\ M) \in R,\ |\alpha| > 2\} \\ & \cup \{(A_{r,i} \to h(\alpha_{i+1})A_{r,i+1};\ \mathbb{N}^3) \mid r = (A \to \alpha;\ M) \in R,\ |\alpha| > 2,\ 2 \le i < |\alpha| - 2\} \\ & \cup \{(A_{r,n-2} \to h(\alpha_{n-1})h(\alpha_n);\ \mathbb{N}^3) \mid r = (A \to \alpha;\ M) \in R,\ n = |\alpha| > 2\} \\ & \cup \{(A_a \to a;\ \mathbb{N}^3) \mid a \in \Sigma\}. \end{aligned}$$

One can easily verify from the definition that G' is in CNF and $L(G') = L(G)$.

G' is in the correct subclasses since all plsets in G' are either unrestricted, or occur in G as well. □

3. Automata

As fledCFG, fledLLING and fledRLING are extensions of context-free resp. left/right linear grammars an obvious idea to find equivalent automata for the new classes is to extend the automata corresponding to the basic classes, which is what we do in this section.

As a result of this approach, most of the proofs we present follow the same ideas and structure as well-known proofs for the basic grammar and automata classes.

3.1. Finite Automata

The general idea behind an equivalence proof of finite automata and right linear grammars is to identify states of the automaton with nonterminal symbols of the grammar. Then, the application of a production in the grammar is equivalent to a state transition in the automaton.

In this bijection, the f-distance at which a production is applied is equal to the position of the automaton's head (given by the number of symbols the automaton has read) before the corresponding state transition. Similarly, the length of the subword generated from the rule application is equal to the number of symbols the automaton will read from this transition to the end of the word.

Formalizing this observation, we find the following definitions and theorems:

Definition 3.1. *A* (nondeterministic) finite automaton with position counter (fedNFA) *is a 5-tuple* $A = (Q, \Sigma, \delta, q_0, F)$*, where*

- Q *is a finite nonempty set of* states*;*
- Σ *is a finite nonempty set called the* input alphabet*;*
- $Q \cap \Sigma = \emptyset$*;*
- $q_0 \in Q$ *is the* initial state*;*
- $F \subseteq Q$ *is the set of* final states*;*
- $\delta\colon\ (Q \times \Sigma_\epsilon \times \mathbb{N}^2) \to \wp Q$ *is called the* transition function*.*

Definition 3.2. *Let* $A = (Q, \Sigma, \delta, q_0, F)$ *be an* fedNFA*. A word* $\kappa \in \Sigma^* Q \Sigma^*$ *is called a* configuration *of A. The relation* $\vdash_A \subseteq (\Sigma^* Q \Sigma^*)^2$ *is defined by* $(u, v \in \Sigma^*,\ a \in \Sigma_\epsilon,\ q, q' \in Q)$*:*

$$uqav \vdash_A uaq'v,\ \text{if } q' \in \delta(q, a, |u|, |av|).$$

The set $T(A) = \{w \in \Sigma^* \mid \exists q_f \in F\colon\ q_0 w \vdash_A^* w q_f\}$ *is called the* language accepted by *A.*
A is called complete*, if* $\forall w \in \Sigma^*\colon\ \exists q \in Q\colon\ q_0 w \vdash_A^* wq$*.*
If A is clear from the context, we will write $\vdash$ *instead of* $\vdash_A$*.*

Intuitively, all values of (f, e) for which $q' \in \delta(q, a, f, e)$ form the set of positions for which the transition from q to q' reading a is allowed, analogous to plsets of fledCFG. Thus, we will refer to these sets as plsets as well.

Just as in the case of grammars, we get subclasses if we do not allow all types of restrictions or force them to be independent.

Definition 3.3. *An* FEDNFA *is called an*

- FENFA *if* $\forall q, q' \in Q, a \in \Sigma_\epsilon\colon\ \exists M_f, M_e \subseteq \mathbb{N}\colon\ \{(f,e) \mid q' \in \delta(q,a,f,e)\} = M_f \times M_e$,
- F[D]NFA *if* $\forall q, q' \in Q, a \in \Sigma_\epsilon\colon\ \exists M_f \subseteq \mathbb{N}\colon\ \{(f,e) \mid q' \in \delta(q,a,f,e)\} = M_f \times \mathbb{N}$,
- E[D]NFA *if* $\forall q, q' \in Q, a \in \Sigma_\epsilon\colon\ \exists M_e \subseteq \mathbb{N}\colon\ \{(f,e) \mid q' \in \delta(q,a,f,e)\} = \mathbb{N} \times M_e$.

As for the grammars, we identify FEDNFA, where each transition is allowed at all positions, equal to the NFA that results from removing these (pseudo-)restrictions and use DNFA as a synonym for that class of automata.

Definition 3.4. [F][E][D]REG *denotes the class of languages accepted by an* [F][E][D]NFA.

Now, we are ready to prove the equivalence of our automata and grammar classes.

Theorem 3.5. [F]E[D]REG = [F]L[D]RLINL *and* FREG = FRLINL.

Proof. Once again, we adapt well-known constructions (cf. [Har78, Chapter 2.5]).

To show [F]E[D]REG ⊆ [F]L[D]RLINL, resp. FREG ⊆ FRLINL, let $A = (Q, \Sigma, \delta, q_0, F)$ an [F][E][D]NFA and set $G = (Q, \Sigma, R, q_0)$, where

$$R = \{(q \to aq';\ \{(f,l) \mid q' \in \delta(q,a,f,l)\} \times \mathbb{N}) \;\big|\; q, q' \in Q,\ a \in \Sigma_\epsilon\} \cup \{(q_f \to \epsilon;\ \mathbb{N}^3) \;\big|\; q_f \in F\}.$$

Obviously G is an [F]L[D]RLING resp. FRLING.

Claim. $\forall u, v \in \Sigma^*,\ q \in Q$:

$$q_0 \dashv\!(0, |uv|, 0)\!\vdash^n uq \ \hookleftarrow\ q_0 uv \vdash^n uqv.$$

Proof. By induction on n.

$n = 0$: We have $q_0 \dashv\!(0, |uv|, 0)\!\vdash^0 uq \ \hookleftarrow\ u = \epsilon \wedge q = q_0 \ \hookleftarrow\ q_0 uv \vdash^0 uqv$.

$n > 0$:
$$\begin{aligned}
& q_0 \dashv\!(0, |uv|, 0)\!\vdash^n u'q' \Rightarrow u'aq = uq \\
\hookleftarrow\ & q_0 \dashv\!(0, |uv|, 0)\!\vdash^n u'q' \wedge \exists M \subseteq \mathbb{N}^3\colon\ (q' \to aq;\ M) \in R \wedge (|u'|, |av|, 0) \in M \\
\hookleftarrow\ & q_0 \dashv\!(0, |uv|, 0)\!\vdash^n u'q' \wedge q \in \delta(q', a, |u'|, |av|) \\
\overset{\text{I.H.}}{\hookleftarrow}\ & q_0 uv \vdash^n u'q'av \wedge q \in \delta(q', a, |u'|, |av|) \\
\hookleftarrow\ & q_0 uv \vdash^n u'q'av \vdash u'aqv = uqv.
\end{aligned}$$
□Claim

From this claim, $L(G) = T(A)$ follows with $uq \Rightarrow w \ \hookleftarrow\ u = w \wedge q \in F$.

To show [F]L[D]RLINL ⊆ [F]E[D]REG resp. FRLINL ⊆ FREG let $G = (N, \Sigma, R, S)$ an [F][L][D]RLING. We may assume w.l.o.g. that the right-hand side of each rule in R contains at most one terminal symbol, and there are no rules that only differ in their plsets. (Rules with longer right-hand sides can be replaced as in the construction of a grammar in CNF in the proof of Lemma 2.16; we mentioned on page 9 how rules that only differ in their plsets can be eliminated.)

Now we can define $A = (N \sqcup \{q_f\},\ \Sigma,\ \delta,\ S,\ \{q_f\})$, where for any $B \in N,\ a \in \Sigma_\epsilon,\ f, e \in \mathbb{N}$:

$$\delta(B, a, f, e) = \{C \;\big|\; (B \to aC;\ M) \in R,\ (f, e, 0) \in M\} \cup \{\, q_f \;\big|\; (B \to a;\ M) \in R,\ (f, e, 0) \in M\}.$$

From this, it follows with Lemma 2.8 that A is an [F]E[D]NFA resp. an FNFA since the plsets of the automaton, $\{(f,e) \mid q' \in \delta(q,a,f,e)\}$, are always the first two components of a plset of the grammar.

Claim. $\forall u, v \in \Sigma^*,\ B \in N$:

$$Suv \vdash^n uBv \ \hookleftarrow\ S \dashv\!(0, |uv|, 0)\!\vdash^n uB.$$

Proof. By induction on n.

$n = 0$: We have $Suv \vdash^0 uBv \;\hookleftarrow\; u = \epsilon \wedge B = S \;\hookleftarrow\; S \dashv(0, |uv|, 0)\vdash\!\Rightarrow^0 uB$.

$n > 0$:
$$\begin{aligned}
& Suv \vdash^n u'B'av \vdash u'aBv = uBv \\
\hookleftarrow\ & Suv \vdash^n u'B'av \wedge B \in \delta(B', a, |u'|, |av|) \\
\hookleftarrow\ & Suv \vdash^n u'B'av \wedge \exists M \subseteq \mathbb{N}^3 \colon\ (B' \to aB;\ M) \in P \wedge (|u'|, |av|, 0) \in M \\
\hookleftarrow\ & Suv \vdash^n u'B'av \wedge u'B' \dashv(0, |uv|, 0)\vdash\!\Rightarrow u'aB \\
\overset{\text{I.H.}}{\hookleftarrow}\ & S \Rightarrow^n u'B' \dashv(0, |uv|, 0)\vdash\!\Rightarrow u'aB = uB
\end{aligned}$$
$\square_{\text{Claim}}$

From this claim $L(G) = T(A)$ follows by applying the same argument as in the induction step to see that $S \Rightarrow^* w'B \Rightarrow w'a = w \hookleftarrow Sw'a \vdash^* w'Ba \vdash w'aq_f = wq_f$. $\square$

Since we have shown in Lemma 2.12 that each language class generated by a class of restricted left-linear grammars is the reversal of one generated by a class of restricted right-linear grammars, we can now relate restricted left-linear language classes to restricted automata classes by showing how the latter behave under reversal.

Lemma 3.6. *For each* FE[D]NFA $A = (Q, \Sigma, \delta, q_0, F)$*, there is an* FE[D]NFA $B = (Q', \Sigma, \delta', q_0', F')$ *with* $T(B) = T(A)^R$*. Furthermore, if A is an* FNFA *then B can be constructed as an* ENFA *and vice versa.*

Proof. We adapt the construction from [AU72, Theorem 2.9] and define B by

- $Q' = Q \sqcup \{q_0'\}$,
- $F' = \{q_0\}$,
- $\forall f, e \in \mathbb{N} \colon\ \delta'(q_0', \epsilon, f, e) = F$ and
- $\forall q \in Q,\ a \in \Sigma_\epsilon,\ f, e \in \mathbb{N} \colon\ \delta'(q, a, f, e) = \{q' | q \in \delta(q', a, e - |a|, f + |a|)\}$.

Then we can show by induction on n that

$$q_0' v^R w^R \vdash_B^{n+1} v^R q w^R \;\hookleftarrow\; \exists q_f \in F \colon\ wqv \vdash_A^n wvq_f.$$

$n = 0$: In this case, the right-hand side enforces $v = \epsilon$.

The claim follows as $\forall q_f \in F,\ w \in \Sigma^* \colon\ q_0' w^R \vdash_B q_f w^R$ and $wq_f \vdash_A^0 wq_f$.

$n > 0$: $q_0' v^R a w^R \vdash_B^{n+1} v^R q' a w^R \vdash_B v^R a q w^R \;\hookleftarrow\; \exists q_f \in F \colon\ wqav \vdash_A waq'v \vdash_A^n wavq_f$,

where the equivalence for the single step follows, since $q \in \delta'(q', a, |v|, |aw|) \hookleftarrow q' \in \delta(q, a, |w|, |av|)$ by the definition of δ', and the equivalence for the remaining computations follows by the induction hypothesis.

From this, we get $T(B) = T(A)^R$ by setting $w = \epsilon$ and $q = q_0$.
The claims on the subclasses follow immediately from the definition of B. $\square$

From Lemmas 2.12, 3.5, and 3.6, we immediately get the following Corollary:

Corollary 3.7.

- FE[D]REG = FL[E][D]RLINL = [F]LE[D]LLINL.
- FREG = F[E][D]RLINL = [F]L[D]LLINL.
- EREG = L[E][D]RLINL = [F]E[D]LLINL.
- REG = [E]RLINL = [F]LLINL.

In the following, we will refer to these as the regular-based language classes and use the respective shorthands as the canonical identifiers for them.

A useful property of traditional finite automata is that every language accepted by an NFA is already accepted by a deterministic finite automaton. This is typically proven constructively by defining the states of the DFA as the powerset of the NFA states and ensuring that the current state of the DFA after reading some input w is the set of all states the NFA can reach after reading w. As we will show now, this construction can be straightforwardly carried over to our extended model.

Definition 3.8. *A* deterministic finite automaton with position counter ([F][E][D]DFA) *is an* [F][E][D]NFA, *satisfying:*

1. $|\delta(q, a, f, e)| \leq 1$ *for all* $q \in Q, a \in \Sigma_\epsilon, f, e \in \mathbb{N}$ *and*
2. *if* $|\delta(q, \epsilon, f, e)| = 1$, *then* $|\delta(q, a, f, e)| = 0$ *for all* $q \in Q, a \in \Sigma, f, e \in \mathbb{N}$.

Theorem 3.9. *For each* [F][E][D]NFA $A = (Q, \Sigma, \delta, q_0, F)$ *there is a complete* [F][E][D]DFA B *without* ϵ*-transitions and with* $T(B) = T(A)$.

Proof. Once more we adapt a known construction, this time from [Neb12, Satz 2.2].

For each $(f, e) \in \mathbb{N} \times \mathbb{N}$, we define the relation $\mathcal{E}'_{(f,e)} \subseteq Q \times Q$ by

$$(q, q') \in \mathcal{E}'_{(f,e)} \rightarrowtail q' \in \delta(q, \epsilon, f, e),$$

and $\mathcal{E}_{(f,e)} = {\mathcal{E}'_{(f,e)}}^*$, that is $\mathcal{E}_{(f,e)}$ contains all the pairs of states (q, q'), so that q' can be reached from q at position (f, e) using only ϵ-transitions (including the pairs (q, q) at any position).

We then define $B' = (Q', \Sigma, \delta', q'_0, F')$ as follows:

- $Q' = \wp Q \sqcup \{q'_0\}$,
- $\forall\ Z \in \wp Q,\ a \in \Sigma,\ f, e \in \mathbb{N}$:
  $\delta'(Z, a, f, e) = \{\{q' \mid \exists q \in Q,\ z \in Z\colon\ q \in \delta(z, a, f, e) \wedge (q, q') \in \mathcal{E}_{(f+1,e-1)}\}\}$
- $\forall a \in \Sigma,\ f, e \in \mathbb{N}$:
  $\delta'(q'_0, a, f, e) = \{\{q' \mid \exists q_1, q_2 \in Q\colon\ (q_0, q_1) \in \mathcal{E}_{(f,e)}) \wedge q_2 \in \delta(q_1, a, f, e) \wedge (q_2, q') \in \mathcal{E}_{(f+1,e-1)}\}\}$,
- $F' = \{Z \in Q' \mid Z \cap F \neq \emptyset\} \cup \{q'_0 \mid \epsilon \in T(A)\}$.

By construction, B' is a complete FEDDFA without ϵ-transitions.

We show by induction on $|u|$ that $\forall u \in \Sigma^+,\ w \in \Sigma^*,\ Z \in \wp Q$:

$$q'_0 uw \vdash^*_{B'} uZw \quad \rightarrowtail \quad Z = \{q | q_0 uw \vdash^*_A uqw\}.$$

$|u| = 1$: Then

$$\begin{aligned} & q'_0 uw \vdash^*_{B'} uZw \\ \rightarrowtail\ & Z = \{q \mid \exists q_1, q_2 \in Q\colon\ (q_0, q_1) \in \mathcal{E}_{(0,|w|+1)} \wedge q_2 \in \delta(q_1, u, 0, |w| + 1) \wedge (q_2, q) \in \mathcal{E}_{(1,|w|)}\} \\ \rightarrowtail\ & Z = \{q \mid q_0 uw \vdash^*_A q_1 uw \vdash_A uq_2 w \vdash^*_A uqw\}. \end{aligned}$$

$|u| > 1$: We can decompose the computation of B' as $q_0' vaw \vdash_{B'}^* vZ'aw \vdash_B uZw$, with $u = va$, $a \in \Sigma$.
By induction hypothesis, the first part is equivalent to $Z' = \{q' \mid q_0 vaw \vdash_A^* vq'aw\}$.
For the second part we find

$$\begin{aligned} & vZ'aw \vdash_{B'} uZw \\ \leftrightarrows\ & Z \in \delta'(Z', a, |v|, |aw|) \\ \leftrightarrows\ & Z = \{q \mid \exists q' \in Q,\ z \in Z' :\ q' \in \delta(z, a, |v|, |aw|) \wedge (q', q) \in \mathcal{E}_{(|u|,|w|)}\} \\ \leftrightarrows\ & Z = \{q \mid \exists q' \in Z' :\ vq'aw \vdash_A^* uqw\} \end{aligned}$$

and the claim follows.

By setting $w = \epsilon$ and applying the definition of F' we get $T(B') = T(A)$.
Now for $Z, Z' \in \wp Q,\ a \in \Sigma$, we have

$$\{(f,e) \mid Z' \in \delta'(Z,a,f,e)\} = \bigcup_{\substack{(q_0,\dots,q_k)\in Q^k \\ q_0\in Z,\ q_k\in Z'}} \Big(\{(f,e) \mid q_1 \in \delta(q_0,a,f,e)\} \cap \bigcap_{1<i\le k} \{(f,e) \mid q_i \in \delta(q_{i+1},\epsilon,f,e)\}\Big).$$

Together with the analogous result for the case $Z = q_0'$ and Lemma 2.8, this shows that if A is an [F][E]DNFA, the proof is complete by setting $B = B'$.

For the case that A is an FENFA we first note that each finite union of FE-plsets and thus each plset of B' is a finite union of *disjoint* FE-plsets, since

$$(M_{f1} \times M_{e1}) \cup (M_{f2} \times M_{e2}) = \big((M_{f1} \cap M_{f2}) \times (M_{e1} \cup M_{e2})\big) \cup \big((M_{f1} \setminus M_{f2}) \times M_{e1}\big) \cup \big((M_{f2} \setminus M_{f1}) \times M_{e2}\big).$$

Let n denote the maximal number of disjoint FE-plsets needed for any transition in B. For each $Z, Z' \in Q'$ and $a \in \Sigma$ and we define sets $M_{f,i}^{(Z,a,Z')}, M_{e,i}^{(Z,a,Z')} \subseteq \mathbb{N}$, $1 \le i \le n$ so that

$$\{(f,e) \mid Z' \in \delta'(Z,a,f,e)\} = \bigsqcup_{1\le i\le n} M_{f,i}^{(Z,a,Z')} \times M_{e,i}^{(Z,a,Z')}.$$

Now we can define $B = (Q'', \Sigma, \delta'', q_0', F'')$ as follows:

- $Q'' = \{q_0'\} \sqcup \{Z^{(i)} \mid Z \in \wp Q,\ 1 \le i \le n\}$,
- $\forall Z^{(i)} \in Q'',\ a \in \Sigma,\ f, e \in \mathbb{N}$:
  $\delta''(Z^{(i)}, a, f, e) = \{Z'^{(j)} \mid Z' \in \delta'(Z,a,f,e),\ f \in M_{f,j}^{(Z,a,Z')},\ e \in M_{e,j}^{(Z,a,Z')}\}$,
- $F'' = \{Z^{(i)} \mid Z \in F'\}$.

It is easily verified that B is an FEDFA and $T(B) = T(B') = T(A)$. □

3.2. Pushdown Automata

While the changes of NFA to fit our extended grammar class are rather straightforward, things are more involved for pushdown automata (PDA). Since each rule application splits the word into three parts, the check if an application is allowed will typically need to combine information from two separate places in the word. As the automaton processes the word in one pass, in order to check if the combination of places is valid, we have to pass information from the first of these places to the second one.

In order to see how this can be implemented, we have a look at the standard construction of a PDA from a grammar (cf. [Har78, Theorem 5.4.1]). The automaton created by this construction works by computing a leftmost derivation on its stack. To this end, the current sentential form is placed on the stack so that its left end is at the top. If the top of stack is a terminal symbol, it is compared to the next symbol of the input word

and if the two symbols match, it is removed from the stack and the automaton moves one character ahead on the input. If the characters do not match, the computation is canceled.

If the top of the stack is a nonterminal, a rule application is simulated by replacing the nonterminal by the (reversed) right-hand side of a rule applicable to it. By construction, the current position of the automaton's head at this point is the f-distance of the rule application. Since its yield is the yield of the symbols just added to the stack, the automaton will have reached the end of this rule application when the first symbol below the newly added right-hand side becomes top of stack. Thus, we can pass information from the beginning to the end of the rule application by putting it on the stack below the symbols from the right-hand side, since this way, the information will become top of stack exactly when it is needed. Specifically, we will put on the stack the current f-distance and the plset, performing the actual check at the end of the simulated rule application.

Formalizing the idea, we get:

Definition 3.10. *A* pushdown automaton with position counter (FLEDPDA) *is a 7-tuple* $A = (Q, \Sigma, \Gamma, \delta, q_0, g_0, F)$*, where*

- Q *is a finite nonempty set of* states*;*
- Σ *is a finite nonempty set called the* input alphabet*;*
- Γ *is a finite nonempty set disjoint of* $(\mathbb{N} \times \wp\mathbb{N}^3)$ *called the* stack alphabet*;*
- $q_0 \in Q$ *is the* initial state*;*
- $g_0 \in \Gamma$ *is the* initial stack symbol*;*
- $F \subseteq Q$ *is the set of* final states*;*
- $\delta\colon\ (Q \times \Sigma_\epsilon \times \Gamma) \to \wp(Q \times \Gamma^* \times \wp\mathbb{N}^3)$ *is called the* transition function*, satisfying* $\forall (q, a, g) \in Q \times \Sigma_\epsilon \times \Gamma\colon\ |\delta(q, a, g)| < \infty$.

Definition 3.11. *Let* $A = (Q, \Sigma, \Gamma, \delta, q_0, g_0, F)$ *an* FLEDPDA*. A word from* $\Sigma^* Q \Sigma^* \times (\Gamma \cup (\mathbb{N} \times \wp\mathbb{N}^3))^*$ *is called a* configuration *of A. The relation* $\vdash_A \subseteq \left(\Sigma^* Q \Sigma^* \times (\Gamma \cup (\mathbb{N} \times \wp\mathbb{N}^3))^*\right)^2$ *is defined by*

$$\begin{aligned} (uqav, \alpha g) &\vdash_A (uaq'v, \alpha(|u|, M)\beta), && \text{if } (q', \beta, M) \in \delta(q, a, g) \text{ and} \\ (uqv, \alpha(f, M)) &\vdash_A (uqv, \alpha), && \text{if } (f, |u| - f, |v|) \in M. \end{aligned}$$

The set $T(A) = \{w \in \Sigma^* \mid \exists q_f \in F,\ \alpha \in (\Gamma \cup (\mathbb{N} \times \wp\mathbb{N}^3))^*\colon\ (q_0 w, g_0) \vdash_A^* (wq_f, \alpha)\}$ *is called the* language accepted by A by final state.

The set $N(A) = \{w \in \Sigma^* \mid \exists q \in Q\colon\ (q_0 w, g_0) \vdash_A^* (wq, \epsilon)\}$ *is called the* language accepted by A by empty stack.

The set $L(A) = \{w \in \Sigma^* \mid \exists q_f \in F\colon\ (q_0 w, g_0) \vdash_A^* (wq_f, \epsilon)\}$ *is called the* language accepted by A by both final state and empty stack.

As before, we will drop the index $_A$ *if it is clear from the context.*

Since FLEDPDA use the same plsets as FLEDCFG the definition of equivalent subclasses is straightforward.

Definition 3.12. *An* FLEDPDA *is called an*

- LEDPDA *if* $\forall (q, a, g) \in Q \times \Sigma_\epsilon \times \Gamma,\ (q', \alpha, M) \in \delta(q, a, g)\colon\ \exists M_{l,e} \subseteq \mathbb{N}^2\colon\ M = \mathbb{N} \times M_{l,e}$,
- FEDPDA *if* $\forall (q, a, g) \in Q \times \Sigma_\epsilon \times \Gamma,\ (q', \alpha, M) \in \delta(q, a, g)\colon$ $\exists M_{f,e} \subseteq \mathbb{N}^2\colon\ M = \{(f, l, e) \mid (f, e) \in M_{f,e},\ l \in \mathbb{N}\}$
- FLDPDA *if* $\forall (q, a, g) \in Q \times \Sigma_\epsilon \times \Gamma,\ (q', \alpha, M) \in \delta(q, a, g)\colon\ \exists M_{f,l} \subseteq \mathbb{N}^2\colon\ M = M_{f,l} \times \mathbb{N}$,
- FLEPDA *if* $\forall (q, a, g) \in Q \times \Sigma_\epsilon \times \Gamma,\ (q', \alpha, M) \in \delta(q, a, g)\colon\ \exists M_f, M_l, M_e \subseteq \mathbb{N}\colon\ M = M_f \times M_l \times M_e$,

- LEPDA *if it is an* LEDPDA *and an* FLEPDA,
- FEPDA *if it is an* FEDPDA *and an* FLEPDA,
- FLPDA *if it is an* FLDPDA *and an* FLEPDA,
- F[D]PDA *if it is an* FLDPDA *and an* FEDPDA,
- L[D]PDA *if it is an* FLDPDA *and an* LEDPDA,
- E[D]PDA *if it is an* LEDPDA *and an* FEDPDA.

Before we prove the equivalence between FLEDPDA and FLEDCFG, we first show that as for traditional PDA, the three modes of acceptance we defined for FLEDPDA are equally powerful.

Theorem 3.13. *For each* [F][L][E][D]PDA $A = (Q, \Sigma, \Gamma, \delta, q_0, g_0, F)$*, there is an* [F][L][E][D]PDA B *with* $L(B) = N(B) = T(B) = T(A)$.

Proof. The basic idea of the construction is that B will simulate A and, whenever A reaches a final state, B has the option of emptying its stack and entering its final state (cf. [Har78, Theorem 5.3.2]).

However, at the end of A's computation, the stack may also contain restrictions that would not have been checked by A but need to be satisfied in order to allow B to empty the stack. To guarantee that these can be removed, we allow B to switch into a mode (the states $\bar{q}$) where it still simulates A, but whenever A places a plset on the stack, B places the unrestricted plset $\mathbb{N}^3$ on the stack instead. To avoid that B uses this to circumvent any check that is performed by A, the guard symbol g_0' is placed atop each of these "disarmed" plsets and we ensure that g_0' can only be removed during the final emptying phase.

Formally, $B = (Q', \Sigma, \Gamma', \delta_B, q_0', \$, F')$, where

- $Q' = \{q, \bar{q} \mid q \in Q\} \sqcup \{q_e, q_f\}$,
- $\Gamma' = \Gamma \sqcup \{\$, g_0'\}$,
- $F' = \{q_f\}$ and
- $$\begin{aligned}
& & \delta_B(q_0', \epsilon, \$) &= \{(q_0, \$g_0, \mathbb{N}^3)\}, \\
\forall (q,a,g) \in Q \times \Sigma \times \Gamma\colon & & \delta_B(q,a,g) &= \delta(q,a,g), \\
\forall (q,a,g) \in Q \times \Sigma \times \Gamma\colon & & \delta_B(\bar{q},a,g) &= \{(\bar{q}', g_0'\alpha, \mathbb{N}^3) \mid \exists M \subseteq \mathbb{N}^3\colon\ (q',\alpha,M) \in \delta(q,a,g)\}, \\
\forall (q,g) \in Q \times \Gamma\colon & & \delta_B(q,\epsilon,g) &= \delta(q,\epsilon,g) \cup \{(\bar{q}, g, \mathbb{N}^3)\} \cup \{(q_e, \epsilon, \mathbb{N}^3) \mid q \in F\}, \\
\forall (q,g) \in Q \times \Gamma\colon & & \delta_B(\bar{q},\epsilon,g) &= \{(\bar{q}', g_0'\alpha, \mathbb{N}^3) \mid \exists M \subseteq \mathbb{N}^3\colon\ (q',\alpha,M) \in \delta(q,\epsilon,g)\} \\
& & & \quad \cup\ \{(q, g, \mathbb{N}^3)\} \cup \{(q_e, \epsilon, \mathbb{N}^3) \mid q \in F\}, \\
\forall q \in F\colon & & \delta_B(q,\epsilon,g_0') &= \{(q_e, \epsilon, \mathbb{N}^3)\}, \\
\forall q \in F\colon & & \delta_B(\bar{q},\epsilon,g_0') &= \{(q_e, \epsilon, \mathbb{N}^3)\}, \\
\forall q \in F\colon & & \delta_B(q,\epsilon,\$) &= \{(q_f, \epsilon, \mathbb{N}^3)\}, \\
\forall q \in F\colon & & \delta_B(\bar{q},\epsilon,\$) &= \{(q_f, \epsilon, \mathbb{N}^3)\}, \\
\forall g \in \Gamma' \setminus \{\$\}\colon & & \delta_B(q_e,\epsilon,g) &= \{(q_e, \epsilon, \mathbb{N}^3)\}, \\
& & \delta_B(q_e,\epsilon,\$) &= \{(q_f, \epsilon, \mathbb{N}^3)\}, \\
\text{otherwise}\colon & & \delta_B(q,a,g) &= \varnothing.
\end{aligned}$$

B is in the same classes as A since each plset used in δ_B is either also used in δ or is $\mathbb{N}^3$. □

Theorem 3.14. *For each* [F][L][E][D]PDA $A = (Q, \Sigma, \Gamma, \delta, q_0, g_0, F)$*, there is an* [F][L][E][D]PDA B *with* $L(B) = N(B) = T(B) = N(A)$.

Proof. Here, the construction from [Har78, Theorem 5.3.1] can be straightforwardly taken over. B places a guard symbol \$ at the bottom of the stack and then simulates A until \$ is the only symbol remaining on the stack. At this point, A would have accepted with empty stack, thus B removes \$ and enters its final state, thereby accepting by all three modes.

Formally $B = (Q', \Sigma, \Gamma', \delta_B, \bar{q_0}, \$, F')$, where

- $Q' = Q \sqcup \{\bar{q_0}, q_f\}$,
- $\Gamma' = \Gamma \sqcup \{\$\}$,
- $F' = \{q_f\}$ and
- $$\begin{array}{rll} & \delta_B(\bar{q_0}, \epsilon, \$) & = \{(q_0, \$g_0, \mathbb{N}^3)\}, \\ \forall (q, a, g) \in Q \times \Sigma_\epsilon \times \Gamma\colon & \delta_B(q, a, g) & = \delta(q, a, g), \\ \forall q \in Q\colon & \delta_B(q, \epsilon, \$) & = \{(q_f, \epsilon, \mathbb{N}^3)\}, \\ \text{otherwise:} & \delta_B(q, a, g) & = \varnothing. \end{array}$$ □

Theorem 3.15. *For each* [F][L][E][D]PDA $A = (Q, \Sigma, \Gamma, \delta, q_0, g_0, F)$*, there is an* [F][L][E][D]PDA B *with* $L(B) = N(B) = T(B) = L(A)$.

Proof. The construction from the previous theorem applies with one small adaption: The transition into q_f as \$ becomes top of stack is only allowed if the current state is in F. □

Now, we show the equivalence of the respective grammar and automaton classes.

Theorem 3.16. *For each* $L \in$ [F][L][E][D]CFL*, there is an* [F][L][E][D]PDA B *with* $L(B) = N(B) = T(B) = L$.

Proof. We will adapt the construction from [Har78, Theorem 5.4.1], as described at the beginning of this section, to get an FLEDPDA A with $N(A) = L$. Then the existence of B follows with Theorem 3.14.

Since $L \in$ [F][L][E][D]CFL, there exists an [F][L][E][D]CFG $G = (N, \Sigma, R, S)$, with $L = L(G)$. Now set $A = (\{q\}, \Sigma, \Gamma = (N \cup \Sigma), \delta, q, S, \{q\})$, where δ is given by

- $\forall A \in N\colon\ \delta(q, \epsilon, A) = \{(q, \alpha^R, M) \mid (A \to \alpha;\ M) \in R\}$,
- $\forall A \in N, a \in \Sigma\colon\ \delta(q, a, A) = \varnothing$,
- $\forall a \in \Sigma\colon\ \delta(q, a, a) = \{(q, \epsilon, \mathbb{N}^3)\},\ \delta(q, \epsilon, a) = \varnothing$.

Clearly, A is an [F][L][E][D]PDA.

We now show by induction over n that for $uvw \in \Sigma^*$, $\gamma \in \Gamma^*$ and $B \in N$,

$$B \dashv\!(|u|, |v|, |w|); L\!\models^n v \quad \rightsquigarrow \quad (uqvw, \gamma B) \vdash^{2n+2|v|} (uvqw, \gamma).$$

From this, $N(A) = L(G)$ follows by setting $u = w = \gamma = \epsilon$ and $B = S$.

$n = 0$: Since $B \neq v$ there is no derivation of the form $B \dashv\!(|u|, |v|, |w|); L\!\models^0 v$.

There also is no computation of the form $(uqvw, \gamma B) \vdash^{2|v|} (uvqw, \gamma)$: Processing the characters of v requires $|v|$ transitions that process one of the characters each and another $|v|$ transitions that remove the plsets put on the stack by the first set of transitions. By the definition of A, none of these transitions removes B.

$n > 0$: Splitting off the first step of the derivation and the first and last step of the computation we find

$$\begin{aligned} & B \Rightarrow \alpha \dashv\!(|u|, |v|, |w|); L\!\models^{n-1} v \\ \rightsquigarrow\ & (B \to \alpha;\ M) \in R,\ (|u|, |v|, |w|) \in M \wedge \alpha \dashv\!(|u|, |v|, |w|); L\!\models^{n-1} v \\ \overset{(*)}{\rightsquigarrow}\ & (B \to \alpha;\ M) \in R,\ (|u|, |v|, |w|) \in M \wedge (uqvw, \gamma(|u|, M)\alpha^R) \vdash^{2n-2+2|v|} (uvqw, \gamma(|u|, M)) \\ \rightsquigarrow\ & (uqvw, \gamma B) \vdash (uqvw, \gamma(|u|, M)\alpha^R) \vdash^{2n-2+2|v|} (uvqw, \gamma(|u|, M)) \vdash (uvqw, \gamma). \end{aligned}$$

The equivalence (*) can be shown by induction on $|\alpha|$:

$|\alpha| = 0$: In this case, $\alpha = v = \epsilon,\ n = 1$, and the claim becomes trivial.

$|\alpha| > 0$: Then, $\alpha = \alpha_1\alpha'$, $\alpha_1 \in (N \cup \Sigma)$, and v can be written as $v = v'v''$, so that the derivation can be split into

$$\alpha_1\alpha' \dashv(|u|, |v|, |w|); L\models^{n_1} v'\alpha' \dashv(|u|, |v|, |w|); L\models^{n-1-n_1} v$$

and the computation can be split into

$$(uqvw, \gamma(|u|, M)\alpha'^R\alpha_1) \vdash^{2n_1+2|v'|} (uv'qv''w, \gamma(|u|, M)\alpha'^R) \vdash^{2n-2-2n_1+2|v''|} (uvqw, \gamma(|u|, M)).$$

The correspondence between the second parts is established by the induction hypothesis of the inner induction. For the first parts, we distinguish two cases:

$\alpha_1 \in \Sigma$: Then $n_1 = 0$, $v' = \alpha_1$, and the correspondence follows since $(q, \epsilon, \mathbb{N}^3) \in \delta(q, \alpha_1, \alpha_1)$ by the definition of A giving (for $\gamma' = \gamma(|u|, M)\alpha'^R$):

$$(uq\alpha_1v''w, \gamma'\alpha_1) \vdash (u\alpha_1qv''w, \gamma'(|u|, \mathbb{N}^3)) \vdash (u\alpha_1qv''w, \gamma').$$

$\alpha_1 \in N$: Then the first part is of the same form as the original claim, but $n_1 < n$. Thus, the induction hypothesis of the outer induction applies. □

For the inverse inclusion, we adapt the construction from [Har78, Theorem 5.4.3].

Theorem 3.17. *For each* [F][L][E][D]PDA $A = (Q, \Sigma, \Gamma, \delta, q_0, g_0, F)$, $L(A) \in$ [F][L][E][D]CFL. *Furthermore if L is in one of the subclasses defined in Definition 3.12, A can be constructed to be of the correspondingly named subtype defined in Definition 2.7.*

Proof. We may assume without loss of generality that $N(A) = T(A) = L(A)$ and $F = \{q_f\}$, since otherwise we can construct A' according to Theorem 3.15 which does satisfy these conditions and accepts the same language.

The basic idea of the construction is that the sequence of sentential forms during a leftmost derivation encodes the sequence of configurations during a computation. The leftmost characters of each sentential form contain the part of the input that has already been processed by the computation as terminal symbols, while the remainder contains the stack contents as nonterminal symbols, the top of stack being leftmost. Additionally, the nonterminal containing top of stack also encodes the currrent state of the PDA.

However, to make this basic idea work we need some additional details. First note that after rule applications that replace a nonterminal by a string from Σ^* (corresponding to transitions that put no symbol on the stack), the top of stack is encoded by a nonterminal that has been added in an earlier step. However, by our idea, this symbol should now encode the current state of the PDA.

To achieve this, we have to encode with each nonterminal the state that will be current when it becomes top of stack. Since we do not know which state this is at the time of the rule application (except for the symbol that is added at the top), we simply add a rule for each possible combination of states. So, for $(q_1, abc) \in \delta(q_2, b, a)$ in an automaton with states $\{q_1, q_2, q_3\}$, we would add the rules

$$\begin{array}{lll}((q_2, a) \to b(q_1, c)(q_1, b)(q_1, a)), & ((q_2, a) \to b(q_1, c)(q_1, b)(q_2, a)), & ((q_2, a) \to b(q_1, c)(q_1, b)(q_3, a)),\\ ((q_2, a) \to b(q_1, c)(q_2, b)(q_1, a)), & ((q_2, a) \to b(q_1, c)(q_2, b)(q_2, a)), & ((q_2, a) \to b(q_1, c)(q_2, b)(q_3, a)),\\ ((q_2, a) \to b(q_1, c)(q_3, b)(q_1, a)), & ((q_2, a) \to b(q_1, c)(q_3, b)(q_2, a)), & ((q_2, a) \to b(q_1, c)(q_3, b)(q_3, a)).\end{array}$$

This way, we are guaranteed that each computation is encoded by a derivation. However, there are also derivations that do not correspond to a computation, namely those where at least one prediction, which state would become current when a symbol becomes top of stack, was wrong.

In order to eliminate these derivations, we add a third component to each nonterminal that repeats the predicted state of the following nonterminal. This can easily be done since we only predict states for nonterminals that are not leftmost in the rule that adds them.

For rules replacing a symbol with a string that contains nonterminals, the rightmost of these nonterminals inherits the third component of the replaced symbol. For rules replacing a symbol with a string without nonterminals, we verify that the new current state matches the third component, and thus the one predicted

for the new top of stack. If it doesn't match, the rule is not added to the grammar, making it impossible to complete the derivation.

Thus the rules for our example above become:

$$\{((q_2, a, q_{i_0}) \to b(q_1, c, q_{i_1})(q_{i_1}, b, q_{i_2})(q_{i_2}, a, q_{i_0})) \mid i_0, i_1, i_2 \in \{1, 2, 3\}\}.$$

Adding position and length restrictions to this construction is straightforward. The yield of a rule application corresponding to a transition is exactly the set of characters consumed while the plset correponding to the transition remains on the stack. Thus, the plset can be used as plset for the rule without changing it.

Formalizing this construction, we define $G = (Q \times \Gamma \times Q, \Sigma, R, (q_0, g_0, q_f))$, where

$$\begin{aligned} R = \{&((q, g, q') \to a;\ M) \mid (q', \epsilon, M) \in \delta(q, a, g)\} \\ &\cup \{((q, g, q') \to a(q_1, \gamma_1, q_2) \cdots (q_{|\gamma|}, \gamma_{|\gamma|}, q_{|\gamma|+1} = q');\ M) \mid \\ &\qquad (q_1, \gamma^R, M) \in \delta(q, a, g),\ q_i \in Q,\ 1 < i \leq |\gamma|\}. \end{aligned}$$

We now show by induction on n that for $g \in \Gamma,\ a \in \Sigma_\epsilon,\ u, v, w \in \Sigma^*$, and $\gamma \in \Gamma^*$,

$$(q, g, q') \dashv[(|u|, |av|, |w|); L]\Rightarrow^n av \quad \rightsquigarrow \quad (uqavw, \gamma g) \vdash^{2n} (uavq'w, \gamma).$$

From this $N(A) = L(G)$ follows by setting $u = w = \gamma = \epsilon$ and $(q, g, q') = (q_0, g_0, q_f)$.

$n = 0$: Since $(q, g, q') \notin \Sigma^*$ there is no such derivation. Since $g \neq \epsilon$ there also is no such computation.

$n = 1$: In this case, $v = \epsilon$ and

$$\begin{aligned} &(q, g, q') \Rightarrow a \\ \rightsquigarrow\ &(q_1, \epsilon, M) \in \delta(q, a, g),\ (|u|, |a|, |w|) \in M \\ \rightsquigarrow\ &(uqaw, \gamma A) \vdash (uaq_1w, \gamma(|u|, M)) \vdash (uaq'w, \gamma). \end{aligned}$$

$n > 1$: Splitting off the first step of the derivation and the first and last step of the computation, we find

$$\begin{aligned} &(q, g, q') \Rightarrow a\alpha = a(q_1, \gamma'_1, q_2) \cdots (q_{|\gamma'|}, \gamma'_{|\gamma'|}, q_{|\gamma'|+1} = q') \dashv[(|u|, |av|, |w|); L]\Rightarrow^{n-1} av \\ \rightsquigarrow\ &(q_1, \gamma'^R, M) \in \delta(q, a, g),\ (|u|, |av|, |w|) \in M \wedge \alpha \dashv[(|ua|, |v|, |w|); L]\Rightarrow^{n-1} v \\ \overset{(*)}{\rightsquigarrow}\ &(q_1, \gamma'^R, M) \in \delta(q, a, g),\ (|u|, |av|, |w|) \in M \wedge (uaq_1vw, \gamma(|u|, M)\gamma') \vdash^{2n-2} (uavq'w, \gamma(|u|, M)) \\ \rightsquigarrow\ &(uqavw, \gamma A) \vdash (uaq_1vw, \gamma(|u|, M)\gamma'^R) \vdash^{2n-2} (uavq'w, \gamma(|u|, M)) \vdash (uavq'w, \gamma). \end{aligned}$$

To show the equivalence (*) we let $k = |\gamma'|$ and split the claim into parts

$$\begin{aligned} &(q_i, \gamma'_i, q_{i+1}) \dashv[(|uv_{1\ldots i-1}|, |v_i|, |v_{i+1\ldots k}w|); L]\Rightarrow^{n_i} v_i \\ \rightsquigarrow\ &(uv_{1\ldots i-1}q_iv_{i\ldots k}w, \gamma(|u|, M)\gamma'_k \cdots \gamma'_i) \vdash^{2n_i} (uv_{1\ldots i}q_{i+1}v_{i+1\ldots k}w, \gamma(|u|, M)\gamma'_k \cdots \gamma'_{i+1}), \end{aligned}$$

$1 \leq i \leq k$, $v_{1\ldots k} = v$. Since each of these claims is of the same form as the original claim, but $n_i < n$, the induction hypothesis applies to them. □

4. Parsing

The results of the previous section suggest that parsing the new language classes can also be done analogous to parsing regular resp. context-free languages, adding checks for position and length at the appropriate points. In this section, we show that this is indeed the case.

In addition to describing the parsing algorithms for FLEDCFG, FLEDRLING, and FLEDLLING we also want to analyze their runtime. Thus, we introduce a measure that captures the time taken for testing the membership of current position and length in the relevant plsets.

Definition 4.1. *For $G = (N, \Sigma, R, S)$ an* FLEDCFG*, we denote by $T_G(n)$ the maximum time over all triples (f, l, e), $f + l + e = n$, needed to compute*

$$R_{(f,l,e)} = \{(A \to \alpha;\ M) \in R \mid (f, l, e) \in M\}.$$

If G is an FLEDRLING *(resp.* FLEDLLING*), we denote by $T'_G(n)$ the maximum time over all pairs (f, e), $f + e = n$, needed to compute*

$$\begin{aligned} R_{(f,e)} &= \{(A \to \alpha;\ M) \in R \mid (f, e, 0) \in M\} \\ &\textit{(resp. } \{(A \to \alpha;\ M) \in R \mid (0, f, e) \in M\}\textit{).} \end{aligned}$$

Obviously, these definitions are only sensible if the sets $R_{(f,l,e)}$ resp. $R_{(f,e)}$ can be computed at all, i.e. if the plsets of the grammar are decidable. We will implicitly assume throughout this chapter, that this is the case.

4.1. Parsing the Regular-Based Language Classes

For the regular-based language classes, parsing can be done efficiently by translating the grammar into a DFA and simulating that on the word to be parsed (cp. [Neb12, Section 2.2.1]).

Theorem 4.2. *If G is an* FLEDRLING *or an* FLEDLLING*, the word problem for G and a word of length n can be solved in runtime in $O(n \cdot T'_G(n))$ after a preprocessing step with a runtime depending on the grammar alone.*

Proof. As shown in Theorems 3.9 and 3.5, there is an FEDDFA A, accepting $L(G)$ and A can be constructed from G in a runtime dependent on the grammar alone.

For each step of A, the new state is determined by the old state, the next symbol of the input and the current position pair (f, e). From the constructions in the aforementioned theorems, it can also be seen that the new state can only differ for two position pairs (f, e) and (f', e') if there is at least one plset that contains $(f, e, 0)$ (resp. $(0, f, e)$ in the case of an FLEDLLING) but not $(f', e', 0)$ (resp. $(0, f', e')$), or vice versa. Thus, instead of using (f, e) directly, the following state can also be determined using the set $R_{(f,e)}$ as a key for lookup and the number of different sets $R_{(f,e)}$ is at most $2^{|R|}$.

Thus, we can precompute a table that is indexed with the current state, the next symbol of input, and the set $R_{(f,e)}$ and yields the new state. The runtime of this computation again only depends on the grammar.

Since A has no ϵ-transitions by the construction of Theorem 3.9, it will in total take exactly n steps. For each step, we have to compute the set $R_{(f,e)}$ (which takes $O(T'_G(n))$ time) and perform a (constant time) table-lookup, adding up to the runtime claimed in the theorem. □

In practice, the computation of the sets $R_{(f,r)}$ might sometimes be significantly sped up by exploiting the fact that they are always required in the order $R_{(0,n)}$, $R_{(1,n-1)}$, $R_{(2,n-2)}$, ..., as computing the changes between neighboring sets might be faster than to compute each set from scratch.

4.2. Parsing the context-free-based Language Classes

For context-free grammars, there are several well-known parsing algorithms. The simplest of those is the CYK algorithm ([Kas65], [You67]) that requires the grammar to be in CNF and takes $O(|R| \cdot n^3)$ time. Valiant's algorithm ([Val75]) computes the same values as CYK (and thus also requires the grammar to be in CNF) using matrix multiplications. Using the fastest matrix multiplication algorithm currently known, this results in a runtime in $O(|R| \cdot n^{2.3727})$ (see [Wil11]). However, the constants involved are so large that in practice, Valiant's algorithm is only faster than the CYK algorithm for very long words. Lastly, Earley's algorithm ([Ear70]) works for any context-free grammar and takes time in $O(\|R\| \cdot n^3)$.

All three parsers belong to the class of chart parsers, that is, in essence they work by filling a two-dimensional chart that (afterwards) contains in entry (i, j) the information which symbols or partial rules can be derived to $w_{i+1,j}$. Since it is straightforward to extract the position and length of a rule application from (i, j) and the length of the word, it comes as no surprise that all three of these parsers can be adapted to FLEDCFG with practically no overhead besides computing the sets $R_{(f,l,r)}$ as the following considerations will show.

4.2.1. CYK Algorithm

When parsing a word w of length n, the CYK algorithm computes a table $\mathbf{T}$ such that

$$\mathbf{T}_{i,j} = \{A \in N \mid A \Rightarrow^* w_{i+1\ldots j}\},\ 0 \le i < j \le n.$$

Then clearly, $w \in L(G) \leftrightarrows S \in \mathbf{T}_{0,n}$.

Since the grammar is in CNF, all the rules are either of the form $(A \to a)$, $a \in \Sigma$ or of the form $(A \to BC)$, $B, C \in N$. Rules of the first kind only contribute to entries $\mathbf{T}_{i,i+1}$, while rules of the second kind only contribute to entries $\mathbf{T}_{i,j}, j > i + 1$.
Thus, we may initialize the table by

for $i = 1 \ldots n$ **do**
  $\mathbf{T}_{i-1,i} := \{A \mid (A \to w_i) \in R\}$;
  for $j = i + 1 \ldots n$ **do**
    $\mathbf{T}_{i-1,j} := \emptyset$;

For the other entries we note that

$$\begin{aligned} A \in \mathbf{T}_{i,j} &\leftrightarrows A \Rightarrow^* w_{i+1\ldots j} \\ &\leftrightarrows \exists B, C \in N,\ i < k < j\colon\ (A \to BC) \in R \wedge B \Rightarrow^* w_{i+1\ldots k} \wedge C \Rightarrow^* w_{k+1\ldots j} \\ &\leftrightarrows \exists B, C \in N,\ i < k < j\colon\ (A \to BC) \in R \wedge B \in \mathbf{T}_{i,k} \wedge C \in \mathbf{T}_{k,j}. \end{aligned}$$

Using this, the following procedure fills $\mathbf{T}$:

for $l = 2 \ldots n$ **do**
  for $i = 0 \ldots n - l$ **do**
    for each $(A \to BC) \in R$ **do**
      for $k = i + 1 \ldots i + l - 1$ **do**
        if $B \in \mathbf{T}_{i,k} \wedge C \in \mathbf{T}_{k,j}$ **then**
          $\mathbf{T}_{i,i+l} := \mathbf{T}_{i,i+l} \cup \{A\}$;
          break for k;

It is obvious that this procedure has a runtime in $O(|R| \cdot n^3)$.
Having filled the table, it is straightforward to extract a leftmost derivation from it using a standard backtracing algorithm. The following algorithm is initially called with parse(0,n,S).

```
parse(i,j,A)
if j = i + 1 then
  print (A → w_i)
else
  for each (A → BC) ∈ R do
    for k = i + 1 ... j − 1 do
      if B ∈ T_{i,k} ∧ C ∈ T_{k,j} then
        print (A → BC);
        parse(i,k,B);
        parse(k,j,C);
        return
```

Again, it is not hard to determine that this procedure has a runtime in $O(|R| \cdot n)$. This means that the filling of the table dominates it.

Now, in order to adapt this algorithm to fledCFG, not much has to change: $A \Rightarrow^* w_{i+1\dots j}$ becomes $A =[(i, j-i, n-j)]\Rightarrow^* w_{i+1\dots j}$, and thus where the original algorithm checks for a rule $(A \to BC) \in R$, we now have to check for $(A \to BC;\ M) \in R$, $(i, j-i, n-j) \in M$ which by the definition of $R_{(f,l,e)}$ is equivalent to $(A \to BC;\ M) \in R_{(i,j-i,n-j)}$. Thus, the following algorithm will fill T in time $O(|R| \cdot n^3 + n^2 \cdot T_G(n))$:

```
for i = 1 ... n do
  Compute R_{(i,1,n−i−1)};
  T_{i−1,i} := {A | (A → w_i; M) ∈ R_{(i,1,n−i−1)}};
  for j = i + 1 ... n do
    T_{i−1,j} := ∅;
for l = 2 ... n do
  for i = 0 ... n − l do
    Compute R_{(i,l,n−l−i)};
    for each (A → BC; M) ∈ R_{(i,l,n−l−i)} do
      for k = i + 1 ... i + l − 1 do
        if B ∈ T_{i,k} ∧ C ∈ T_{k,j} then
          T_{i,i+l} := T_{i,i+l} ∪ {A};
          break for k;
```

The adaptation of the backtracing algorithm is equally straightforward: Simply replace R by $R_{(i,j-i,n-j)}$.

4.2.2. Valiant's Algorithm

As mentioned before, Valiant's algorithm computes the same table $\mathbf{T}$ as the CYK algorithm. It uses a different approach to do so, however.

In order to reduce parsing to matrix multiplication, Valiant first defines an operation $*$ on the possible entries of $\mathbf{T}$, i.e. on $\wp N$ by

$$\mathbf{S} * \mathbf{U} = \{A \mid (A \to BC) \in R \land B \in \mathbf{S} \land C \in \mathbf{U}\}.$$

Using this operation as multiplication and union as addition he then defines a matrix product by

$$(\mathbf{S} * \mathbf{U})_{i,j} = \bigcup_{0 \le k \le n} \mathbf{S}_{i,k} * \mathbf{U}_{k,j}$$

and the transitive closure of a matrix by

$$\mathbf{S}^+ = \bigcup_{i \ge 1} \mathbf{S}^{(i)},$$

where $\mathbf{S}^{(1)} = \mathbf{S}$ and

$$\mathbf{S}^{(i)} = \bigcup_{0<j<i} \mathbf{S}^{(j)} * \mathbf{S}^{(i-j)},\ i > 1.$$

Using these definitions and letting $\mathbf{D}$ denote the matrix constructed by the initialization step of the CYK algorithm, we have $\mathbf{T} = \mathbf{D}^+$. Thus, parsing is reduced to computing the transitive closure of a triangular matrix.

Following this, Valiant gives a recursive procedure that computes the transitive closure of a triangular matrix using the matrix product as elementary operation. The runtime of this procedure is in $O(\max\{M_*(n), n^2 \log n\})$, where $M_*(n)$ is the time required to compute the $*$-product of two $n \times n$ matrices.

Finally, Valiant shows that the $*$-product of two $n \times n$ matrices $\mathbf{V} = \mathbf{S} * \mathbf{U}$ can be computed using $|R|$ multiplications of $n \times n$ binary matrices in the following way:

For each symbol $A \in N$, define $\mathbf{S}_A$ by

$$(\mathbf{S}_A)_{i,j} = \begin{cases} 1 & A \in \mathbf{S}_{i,j} \\ 0 & A \notin \mathbf{S}_{i,j} \end{cases},$$

and $\mathbf{U}_A$ analogously. Then, for each $r = (A \to BC) \in R$ compute $\mathbf{V}_r = \mathbf{S}_B \cdot \mathbf{U}_C$, and for each $A \in N$, compute

$$\mathbf{V}_A = \bigvee_{r \in R_A} \mathbf{V}_r.$$

Finally,

$$\mathbf{V}_{i,j} = \{A \mid (\mathbf{V}_A)_{i,j} = 1\}.$$

It is easily verified that $\mathbf{V} = \mathbf{S} * \mathbf{U}$.

Combined, this gives a runtime in $O(\max\{|R| \cdot M(n), |R| \cdot n^2 \log n\})$ for the complete table filling algorithm, where $M(n)$ is the runtime to compute the product of two $n \times n$ binary matrices. This still dominates the runtime for the backtrace, which can of course be carried over from the CYK algorithm.

Switching to FLEDCFG, we once again have to eliminate rules from contributing to matrix entries corresponding to positions/lengths where they are not allowed.

This can not be done by just altering the product of two matrix entries as the new version

$$\mathbf{S} * \mathbf{U} = \{A \mid (A \to BC;\ M) \in R_{(i,j-i,n-j)} \wedge B \in \mathbf{S} \wedge C \in \mathbf{U}\}$$

now depends on which matrix entry it contributes to.

We may, however, use the modified product to define a matrix product directly by

$$(\mathbf{S} * \mathbf{U})_{i,j} = \bigcup_{k=0}^{n} \{A \mid (A \to BC;\ M) \in R_{(i,j-i,n-j)} \wedge B \in \mathbf{S}_{i,k} \wedge C \in \mathbf{U}_{k,j}\}.$$

The definition of transitive closure then remains as before and a straightforward induction shows that $\mathbf{T} = \mathbf{D}^+$ still holds.

Now, a careful inspection of Valiant's algorithm shows that each product of two submatrices computed by the algorithm contributes to only one submatrix of the result and the coordinates of this submatrix are known at the time of computation. Thus, the algorithm also works with the modified matrix product.

What remains to be done is to compute $\mathbf{V} = \mathbf{S} * \mathbf{U}$, where $\mathbf{V}_{0,0}$ contributes to $\mathbf{T}_{i',j'}$ (and the remainder of $\mathbf{V}$ accordingly) using multiplication of binary matrices. This can be done by a relatively straightforward modification of Valiant's procedure:

$\mathbf{S}_A$ and $\mathbf{U}_A$ are defined as before. For each $r = (A \to BC) \in R$, compute $\mathbf{V}_{r,1} = \mathbf{S}_B \cdot \mathbf{U}_C$ and $\mathbf{V}_{r,2}$, defined by

$$(\mathbf{V}_{r,2})_{i,j} = \begin{cases} 1 & r \in R_{(i+i',j+j'-i-i',n-j-j')} \\ 0 & r \notin R_{(i+i',j+j'-i-i',n-j-j')} \end{cases}.$$

Then $\mathbf{V}_r = \mathbf{V}_{r,1} \odot \mathbf{V}_{r,2}$, where $\odot$ is component-wise product, and from this, $\mathbf{V}_A$ and $\mathbf{V}$ are computed as before. It is again easily verified that this computes the desired values. Since the runtime of the additional operations is also dominated by the matrix multiplications we get the following theorem:

Theorem 4.3. *If G is an* FLEDCFG *in CNF, the word problem for G and a word of length n can be solved in runtime in $O(\max\{|R| \cdot M(n), |R| \cdot n^2 \log n, n^2 \cdot T_G(n)\})$, where $M(n)$ is the runtime to compute the product of two $n \times n$ binary matrices.* □

4.2.3. Earley's Algorithm

As the other two algorithms, Earley's algorithm fills a triangular matrix $\mathbf{T}$. However, instead of lists of nonterminals, the entries now consist of dotted productions $(A \to \alpha \bullet \beta)$, where $(A \to \alpha\beta) \in R$. We call a dotted production $(A \to \alpha \bullet \beta)$ a valid entry for $\mathbf{T}_{i,j}$ if and only if

$$\alpha \Rightarrow^* w_{i+1\ldots j} \ \wedge \ S \Rightarrow^* w_{1\ldots i} A\gamma,$$

for some $\gamma \in (N \cup \Sigma)^*$. Then if the table contains exactly the valid items we have

$$w \in L(G) \ \leftrightarrows \ (S \to \alpha\bullet) \in \mathbf{T}_{0,n},$$

for any rule with S on the left-hand side.

As can easily be seen, the correctness of the conclusion $w \in L(G)$ only depends on the first condition for a valid entry $\alpha \Rightarrow^* w_{i+1\ldots j}$. Thus the parser would also be correct if the second condition were weakened or dropped, assuming we change its operation to satisfy the new condition. The advantage of keeping the condition is that for most grammars, the number of items that has to be computed is reduced.

Since $\epsilon \Rightarrow^* \epsilon$ and $S \Rightarrow^* S = w_{0\ldots 0} S \epsilon$, we may initialize the table by setting

$$\mathbf{T}_{0,0} = \{(S \to \bullet\alpha) \mid (S \to \alpha) \in R\}.$$

Now, the algorithm in essence computes the closure of the initialized table under a set of three operations:

Scanner: When an entry $(A \to \alpha \bullet a\beta) \in \mathbf{T}_{i,j}$ is present and $a = w_{j+1}$, add $(A \to \alpha a \bullet \beta)$ to $\mathbf{T}_{i,j+1}$.

Predictor: When an entry $(A \to \alpha \bullet B\beta) \in \mathbf{T}_{i,j}$, $B \in N$, is present, add $(B \to \bullet\gamma)$ to $\mathbf{T}_{j,j}$ for each $(B \to \gamma) \in R_B$.

Completer: When entries $(A \to \alpha \bullet B\beta) \in \mathbf{T}_{i,k}$ and $(B \to \gamma\bullet) \in \mathbf{T}_{k,j}$ are present, add $(A \to \alpha B \bullet \beta)$ to $\mathbf{T}_{i,j}$.

It is straightforward to verify that each of these operations adds only valid entries to the table, if the entries they start with are valid:

Scanner: $(A \to \alpha \bullet a\beta) \in \mathbf{T}_{i,j}$ implies $\alpha \Rightarrow^* w_{i+1\ldots j}$. By adding $a = w_{j+1}$ to the right of each sentential form in this partial derivation we get $\alpha a \Rightarrow^* w_{i+1\ldots j+1}$. Since the condition $S \Rightarrow^* w_{1\ldots i} A\gamma$ is identical for both entries, it is inherited.

Predictor: $\epsilon \Rightarrow^* \epsilon$ is immediate. $\alpha \Rightarrow^* w_{i+1\ldots j}$, $S \Rightarrow^* w_{1\ldots i} A\gamma$, and the rule $(A \to \alpha B\beta)$ can be combined to $S \Rightarrow^* w_{1\ldots i} A\gamma \Rightarrow w_{1\ldots i}\alpha B\beta\gamma \Rightarrow^* w_{1\ldots j} B\beta\gamma$.

Completer: For the first condition, $\alpha \Rightarrow^* w_{i+1\ldots k}$, $(B \to \gamma) \in R$, and $\gamma \Rightarrow^* w_{k+1\ldots j}$ can be combined to $\alpha B \Rightarrow^* w_{i+1\ldots k} B \Rightarrow w_{i+1\ldots k}\gamma \Rightarrow^* w_{i+1\ldots j}$. As in the Scanner, the condition $S \Rightarrow^* w_{1\ldots i} A\gamma$ can be inherited from $(A \to \alpha \bullet B\beta) \in \mathbf{T}_{i,k}$.

From these considerations, we also see that scanner and completer verify the first condition, simply passing through the second one, while the predictor checks the second condition, the first one automatically being true for each predicted rule. This implies that if we were to weaken the second condition, we would only have to adapt the predictor.

Now, after verifying that the algorithm only adds valid items, we will show it adds all of them. To this end, assume to the contrary there are valid entries that are not added by the algorithm. Among those, let $(A \to \alpha x \bullet \beta) \notin \mathbf{T}_{i,j}$ with either $x \in (N \cup \Sigma)$ or $\alpha x = \epsilon$, selected as follows:

1. From all missing items, select those where j is minimal.
2. From those items with minimal j, select the ones where the corresponding partial derivation $S \Rightarrow^* w_{1\ldots i}A\gamma \Rightarrow^* w_{1\ldots j}\beta\gamma$ has the fewest steps in both parts combined.
3. From these, select the ones where the number of steps in the second part is lowest.
4. If there are still multiple candidates select an arbitrary one.

Now, we distinguish three cases:

$x \in \Sigma$: Then, $\alpha x \Rightarrow^* w_{i+1\ldots j}$ implies $x = w_j$ and $\alpha \Rightarrow^* w_{i+1\ldots j-1}$. Thus $(A \to \alpha \bullet x\beta)$ is a valid entry for $\mathbf{T}_{i,j-1}$ and since it has not been chosen instead of $(A \to \alpha x \bullet \beta)$ we can conclude from $j - 1 < j$ that $(A \to \alpha \bullet x\beta) \in \mathbf{T}_{i,j-1}$. However, this and $x = w_j$ imply that the scanner would have added $(A \to \alpha x \bullet \beta)$ to $\mathbf{T}_{i,j}$. So this case is impossible.

$x \in N$: Then, $\alpha x \Rightarrow^* w_{i+1\ldots j}$ implies the existence of $k \leq j$ such that $\alpha \Rightarrow^* w_{i+1\ldots k}$ and $x \Rightarrow^* w_{k+1\ldots j}$. The first derivation implies that $(A \to \alpha \bullet x\beta)$ is a valid entry for $\mathbf{T}_{i,k}$. Since the second of the derivations contains at least one step, the derivation $S \Rightarrow^* w_{1\ldots i}A\gamma \Rightarrow^* w_{1\ldots k}x\beta\gamma$ contains fewer steps than the derivation $S \Rightarrow^* w_{1\ldots i}A\gamma \Rightarrow^* w_{1\ldots j}\beta\gamma$. Since also $k \leq j$, we can conclude that $(A \to \alpha \bullet x\beta) \in \mathbf{T}_{i,k}$ since otherwise it would have been chosen instead of $(A \to \alpha x \bullet \beta)$.

Furthermore, $x \Rightarrow^* w_{k+1\ldots j}$ and $S \Rightarrow^* w_{1\ldots j}\beta\gamma$ imply that $(x \to \alpha'\bullet)$ is valid for $\mathbf{T}_{k,j}$, where $(x \to \alpha')$ is the first rule used in $x \Rightarrow^* w_{k+1\ldots j}$. Since furthermore, $S \Rightarrow^* w_{1\ldots i}A\gamma \Rightarrow^* w_{1\ldots j}\beta\gamma$ and $S \Rightarrow^* w_{1\ldots k}x\beta\gamma \Rightarrow^* w_{1\ldots j}\beta\gamma$ are the same derivation but the latter representation has fewer steps in the second part we get that $(x \to \alpha'\bullet) \in \mathbf{T}_{k,j}$ since otherwise it would have been chosen instead of $(A \to \alpha x \bullet \beta)$. However, if $(A \to \alpha \bullet x\beta) \in \mathbf{T}_{i,k}$ and $(x \to \alpha'\bullet) \in \mathbf{T}_{k,j}$, the completer would have added $(A \to \alpha x\beta)$ to $\mathbf{T}_{i,j}$. So we can rule out this case.

$\alpha x = \epsilon$: Then $j = i$, and $S \Rightarrow^* w_{1\ldots i}A\gamma$ implies $S \Rightarrow^* w_{1\ldots i'}A'\gamma'$ and $A' \Rightarrow^* w_{i'+1\ldots i}A\gamma''$, where $\gamma = \gamma''\gamma'$ and A is introduced in the first step of the second derivation. Furthermore, for $(A' \to \alpha' A\beta')$ the rule used in this first step, we get that $(A' \to \alpha' \bullet A\beta')$ is a valid entry for $\mathbf{T}_{i',i}$. Now, since $S \Rightarrow^* w_{1\ldots i}A\gamma$ contains fewer steps than $S \Rightarrow^* w_{1\ldots i}A\gamma \Rightarrow w_{1\ldots i}\beta\gamma$ and $(A' \to \alpha' \bullet A\beta')$ has not been chosen, we can conclude that $(A' \to \alpha' \bullet A\beta') \in \mathbf{T}_{i',i}$. From this however, the predictor would have added $(A \to \bullet\beta)$ to $\mathbf{T}_{i,j}$, contradicting this case.

Since these cases cover all possibilities for x, we can conclude that Earley's algorithm is correct.

If we define $\|R\| = \sum_{(A\to\alpha)\in R}(|\alpha| + 1)$, the filled table will have $O(\|R\| \cdot n^2)$ entries. In an efficient implementation, each operation will consider each entry resp. pair of entries at most once to determine which entries are to be added based on it.

Since the scanner will add exactly one new entry for each eligible entry it finds, it will contribute $O(\|R\| \cdot n^2)$ to the total runtime.

The items added by the predictor only depend on the nonterminal that follows after the point in the entry already present and the column this entry belongs to. Thus, a clever implementation will maintain a list of nonterminals that have been predicted for each column and each invocation of the predictor first checks if the given nonterminal has already been predicted for the column at hand. Only if this is not the case, it adds the new entries. The $O(\|R\| \cdot n^2)$ checks dominate the $O(|R| \cdot n)$ additions of new items and both operations take constant time per invocation.

Similarly to the predictor, the completer can only add new entries based on $(B \to \gamma\bullet) \in \mathbf{T}_{k,j}$ if no entry $(B \to \gamma'\bullet) \in \mathbf{T}_{k,j}$ has been processed before. Thus, we can ensure that each possible entry in $\mathbf{T}_{i,j}$ will be added at most once for each k, $i \leq k \leq j$. This way, we can bound the runtime of the completer to $O(\|R\| \cdot n^2)$ checks and $O(\|R\| \cdot n^3)$ additions, again taking constant time each.

We complete the parsing algorithm by giving a backtracing procedure for the table filled by Earley's algorithm. It is initially called with parse(0,n,$(S \to \alpha\bullet)$) for $(S \to \alpha\bullet)$ an item in $\mathbf{T}_{0,n}$.

parse(i,j,($A \to \alpha C \bullet \beta$))
for each ($C \to \gamma$) $\in R$ **do**
  for $k = i+1 \dots i+l-1$ **do**
    if ($A \to \alpha \bullet C\beta$) $\in \mathbf{T}_{i,k} \wedge$ ($C \to \gamma\bullet$) $\in \mathbf{T}_{k,j}$ **then**
      if $\alpha = \epsilon$ **then**
        print ($A \to \alpha C\beta$);
      else
        parse(i,k,($A \to \alpha \bullet C\beta$));
      parse(k,j,($C \to \gamma\bullet$));
      return

As for its CYK counterpart, the runtime of this procedure is in $O(|R| \cdot n)$.

Combining the times for all four operations, we find that the parser runs in $O(\|R\| \cdot n^3)$ time. Notes on an efficient implementation can for example be found in [GHR80] or [Sto94].

In order to adapt the algorithm to FLEDCFG, the natural adaptation of the condition for $(A \to \alpha \bullet \beta) \in \mathbf{T}_{i,j}$ is

$$\alpha \;=\!\![(i, j-i, n-j)]\!\!\Rightarrow^* w_{i+1\dots j} \;\wedge\; S \;=\!\![(0,n,0)]\!\!\Rightarrow^* w_{1\dots i}A\gamma. \tag{4.1}$$

Looking at the three parser operations, we note that the scanner will be correct for the adapted first condition without any change.

The completer for CFG makes use of the fact that $(B \to \gamma\bullet) \in \mathbf{T}_{k,j}$ not only implies $\gamma \Rightarrow^* w_{k+1\dots j}$ but also $(B \to \gamma) \in R$, and thus $B \Rightarrow^* w_{k+1\dots j}$. Thus $\alpha B\beta \Rightarrow^* w_{i+1\dots k}B\beta$ can be extended to $\alpha B\beta \Rightarrow^* w_{i+1\dots j}\beta$. For FLEDCFG, the extension from $\gamma \;=\!\![(k, j-k, n-j)]\!\!\Rightarrow^* w_{k\dots j}$ to $B \;=\!\![(k, j-k, n-j)]\!\!\Rightarrow^* w_{k\dots j}$ using $(B \to \gamma;\ M)$ is only allowed if $(k, j-k, n-j) \in M$. As this is equivalent to $(B \to \gamma;\ M) \in R_{(k,j-k,n-j)}$, we need to check this property on each completion. This check can be done in $O(1)$ time once we have computed $R_{(k,j-k,n-j)}$.

Looking at the second condition from Equation (4.1), we note that it implies that all the rules used in the partial derivation from S to $w_{1\dots i}A\gamma$ are applied with an allowed position and length. However, as we have just seen that it is convenient to check the lengths and positions in the completer, the parser will only have verified this property for rule applications whose yield lies within $w_{1\dots i}$.

To work around this, we make use of the fact that we may weaken the second condition without affecting the correctness of the algorithm if we adapt the predictor to the weakened condition, as we have seen above. But instead of adapting the predictor to the strong condition (4.1), we may weaken the condition to match the actual operation of the predictor.

If we denote by G' the CFG that results from G by removing all plsets, the appropriate condition is

$$\begin{aligned}&(A \to \alpha \bullet \beta) \in \mathbf{T}_{i,j}\\ \leftrightsquigarrow\;& \alpha \;=\!\![(i, j-i, n-j)]\!\!\Rightarrow^* w_{i+1\dots j} \;\wedge\; \exists\delta : (S \Rightarrow^*_{G'} \delta A\gamma \wedge \delta \;=\!\![(0, i, n-i)]\!\!\Rightarrow^* w_{1\dots i}).\end{aligned}$$

Using this condition, the correctness proof from CFG can now be carried over by adapting notation and including above remarks.

Since the only additional operation that has a notable effect on the runtime is the computation of the sets $R_{(f,l,r)}$ we get the following result:

Theorem 4.4. *If G is an* FLRDCFG*, the word problem for G and a word of length n can be solved in runtime in* $O(\max\{\|R\| \cdot n^3, n^2 \cdot T_G(n)\})$. □

5. Hierarchy of Language Classes

Continuing in the spirit of the previous sections, we are able to give a generalization of the Myhill-Nerode Theorem that allows us to decide membership for most of the regular-based language classes.

Definition 5.1. *Let Σ be an alphabet, $L \subseteq \Sigma^*$. For $f, e \in \mathbb{N} \cup \{*\}$, we define $R_{L,(f,e)} \subseteq \Sigma^f \times \Sigma^f$ by*

$$(x, y) \in R_{L,(f,e)} \quad \rightarrowtail \quad \forall z \in \Sigma^e : (xz \in L \rightarrowtail yz \in L).$$

Theorem 5.2 ([Ner58])**.** *L is regular $\rightarrowtail$ the set of equivalence classes $\Sigma^*/R_{L,(*,*)}$ is finite.* □

Theorem 5.3.

- $L \in$ fedREG $\rightarrowtail \exists n \in \mathbb{N} : \forall (f, e) \in \mathbb{N}^2 : \ |\Sigma^f / R_{L,(f,e)}| \leq n.$
- $L \in$ fREG $\rightarrowtail \exists n \in \mathbb{N} : \forall f \in \mathbb{N} : \ |\Sigma^f / R_{L,(f,*)}| \leq n.$
- $L \in$ eREG $\rightarrowtail \exists n \in \mathbb{N} : \forall e \in \mathbb{N} : \ |\Sigma^* / R_{L,(*,e)}| \leq n.$

Proof. We adapt the proof of Theorem 5.2. First assume $L \in$ fedREG. Then, there is an fedDFA $A = (Q, \Sigma, \delta, q_0, F)$ with $T(A) = L$. Now, for any $q \in Q,\ v \in \Sigma^e$ let $u_1, u_2 \in \Sigma^f$ so that $q_0 u_i v \vdash_A^* u_i q v, i \in \{1, 2\}$. Then since $|u_1| = |u_2|$, both computations will continue identically. Thus, $u_1 v \in L \rightarrowtail u_2 v \in L$. Additionally, for any $v' \in \Sigma^e$ we also have $q_0 u_i v' \vdash_A^* u_i q v'$, $i \in \{1, 2\}$ and thus $u_1 v' \in L \rightarrowtail u_2 v' \in L$. This gives $(u_1, u_2) \in R_{L,(f,e)}$. Hence, $R_{L,(f,e)}$ induces at most $n = |Q|$ equivalence classes.

For the other direction, assume that $\forall (f, e) \in \mathbb{N}^2 :\ R_{L,(f,e)}$ induces n equivalence classes $C_{1,(f,e)}, \ldots, C_{n,(f,e)}$; if there are fewer actual classes, we may fill up with empty ones. Define

$$A = (\{q_0, \ldots, q_n, q_{-1}\},\ \Sigma,\ \delta,\ q_0,\ \{q_{-1}\}),$$

where δ is given by

$$\begin{aligned}
&\delta(q_0, \epsilon, 0, e) = \{q_i \mid \epsilon \in C_{i,(0,e)}\},\ e \in \mathbb{N},\\
&\delta(q_i, a, f, e) = \{q_j \mid \exists u \in C_{i,(f,e)} :\ ua \in C_{j,(f+1,e-1)}\},\ a \in \Sigma,\ f \in \mathbb{N},\ e \in \mathbb{N}_+\\
&\delta(q_i, \epsilon, f, 0) = \{q_{-1} \mid C_{i,(f,0)} \cap L \neq \varnothing\},\ f \in \mathbb{N}.
\end{aligned}$$

In order to show that A is an fedNFA, we need to show that it is well defined. This is obvious for the first and third type of transition. For the second one, assume to the contrary that there exist $u_1, u_2 \in C_{i,(f,e)},\ a \in \Sigma$, so that

$$u_1 a \in C_{j_1,(f+1,e-1)} \neq C_{j_2,(f+1,e-1)} \ni u_2 a.$$

Then, $\exists z \in \Sigma^{e-1} :\ u_1 a z \in L \wedge u_2 a z \notin L$ or vice versa. But since $az \in \Sigma^e$, this implies $(u_1, u_2) \notin R_{L,(f,e)}$, contradicting the assumption that they are in the same equivalence class.

A similar argument shows that $C_{i,(f,0)} \cap L \neq \varnothing$ implies $C_{i,(f,0)} \subseteq L$. From this we get $T(A) = L$. Thus $L \in$ fedREG.

The proofs for fREG and eREG are obtained from this by replacing any instance of e resp. f by $*$ and adapting some technical details. □

We did not include feREG in the Theorem as there does not seem to be a useful characterization of that class in terms of equivalence classes. In order to see this, consider an arbitrary feDFA and for each pair (f, e) split Σ^f into equivalence classes as induced by that automaton. Now, change the labelling/numbering of the classes for some of the pairs (f, e), e.g. those for which $f + e$ is prime and construct the fedNFA from the

relabelled classes. You will observe that typically the resulting automaton is not an feNFA. This shows that the property of being in feREG is not inherent in the equivalence classes induced by a language.

Given this observation, we can only resort to the characterisation that a language is in feREG if it is in fedREG and there is a way of labelling the equivalence classes so that the construction in the proof of Theorem 5.3 happens to yield an feNFA.

Switching to CFG, a result that can be carried over immediately from the traditional setting is the Interchange Lemma.

Lemma 5.4 (Interchange Lemma; [ORW85])**.** *Let $L \in$ fledCFL. Then, there is a constant $c_L \geq 1$ such that for any integer $n \geq 2$, any subset Q_n of $L_n = \{w \in L \mid |w| = n\}$ and any integer m with $n \geq m \geq 2$ there are $k \geq |Q_n|/c_L n^2$ words $z_i \in Q_n$ with the following properties:*

1. $z_i = w_i x_i y_i,\ i = 1, \ldots, k$;
2. $|w_1| = |w_2| = \cdots = |w_k|$;
3. $|y_1| = |y_2| = \cdots = |y_k|$;
4. $m \geq |x_1| = |x_2| = \cdots = |x_k| > \frac{m}{2}$;
5. $w_i x_j y_i \in L_n$ *for all* $i, j \in \{1, \ldots, k\}$.

Proof. The proof from [ORW85] can be carried over immediately:

There is an fledCFG G in CNF with $L = L(G)$. Let $n, m \in \mathbb{N}$ with $n \geq m \geq 2$. We show that

$$\forall z \in L_n(G)\colon\ \exists A \in N\colon\ S \Rightarrow^* wAy \Rightarrow^* wxy = z \wedge m \geq |x| > \frac{m}{2}. \tag{5.1}$$

Since G is in CNF each node in a parse-tree for z has degree at most 2. Thus, if a node is the root of a subtree with more than m leaves, at least one of its children is the root of a subtree with more than $\frac{m}{2}$ leaves. By starting at the root and always descending to the child with the larger subtree we will eventually arrive at a node which is the root of a subtree with at most m and more than $\frac{m}{2}$ leaves. Setting A to the label of this node and x to the subword generated in its subtree we get the derivation claimed in Equation (5.1).

Now, for each $A \in N,\ n_1, n_2 \in \mathbb{N}$ with $n_1 + n_2 \leq n$, we define

$$Q_{n_1,n_2}(n, A) = \{z \in Q_n \mid S \Rightarrow^* wAy \Rightarrow^* wxy = z \wedge |w| = n_1 \wedge |y| = n_2\}.$$

From the definition of $Q_{n_1,n_2}(n, A)$ and Equation (5.1), we get that

$$Q_n = \bigcup_{\substack{A \in N \\ n_1, n_2 \in \mathbb{N} \\ m \geq n - n_1 - n_2 > \frac{m}{2}}} Q_{n_1,n_2}(n, A).$$

Since furthermore for $n \geq m \geq 2$, some basic algebra shows that

$$\left|\left\{(n_1, n_2) \in \mathbb{N}^2 \;\middle|\; m \geq n - n_1 - n_2 > \frac{m}{2}\right\}\right| < n^2$$

holds, we can conclude that

$$\forall Q_n \subseteq L_n(G)\colon\ \exists n_1, n_2 \in \mathbb{N},\ A \in N\colon\ m \geq n - n_1 - n_2 > \frac{m}{2} \wedge |Q_{n_1,n_2}(n, A)| \geq \frac{|Q_n|}{|N| n^2}.$$

From this the lemma now follows by setting $c_L = |N|$, since if $z_1, z_2 \in Q_{n_1,n_2}(n, A)$ with derivations $S \Rightarrow^* w_i A y_i \Rightarrow^* w_i x_i y_i = z_i,\ i \in \{1, 2\}$, we can combine these to a derivation $S \Rightarrow^* w_1 A y_1 \Rightarrow^* w_1 x_2 y_1$. This is a valid derivation in G since every rule in the first part is applied with the same length and position as in the derivation of z_1 and every rule in the second part is applied with the same length and position as in the derivation of z_2. ◻

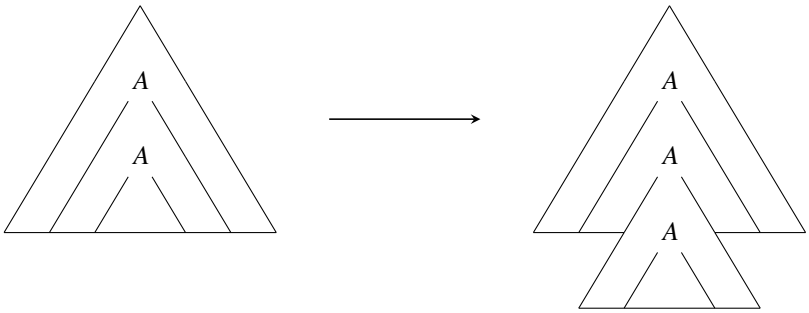


Figure 5.1.: The pumping lemma for context-free languages.

Now the next thing to look at are the various pumping lemmas for regular or context-free languages. This, however, is where our lucky streak of easily transformable results ends.

Figure 5.1 depicts the basic idea of the pumping lemma for context-free languages: If a derivation contains a subtree that has a root labelled with A and another internal node also labelled with A then the partial derivation represented by the partial tree between those two nodes can be repeated arbitrarily often.

This no longer works for FLEDCFG since the insertion (or deletion) of such partial trees changes the lengths and/or positions of rule applications and we are not guaranteed that these new positions and lengths are allowed. For example the f- and e-distance of any rule in the subtree below the pumped part will increase with each copy of the pumped part, as will the length of the rule applied at the root of the tree.

Since every FLEDCFG is in essence a CFG with some derivations/words filtered out, one might get the idea that in order to get an infinite language we should at least get a word from the language at hand for infinitely many of the pumped words $uv^i wx^i y$.

However, this idea does not take into account the fact that there may be multiple pumping sites in a word, and the pumped word will only be in the language if the number of times it is pumped at each site satisfies some relation. This happens, e.g., for the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, since each pumping site may pump at most two of the letters in order for the pumped word to still be of the form $a^* b^* c^*$.

Since the relations between different pumping sites may be arbitrarily complex (think of languages $a^n b^{f(n)}$ for arbitrary f), it is unlikely that the concept of the pumping lemma can be carried over to FLEDCFG.

While this means that we do not have any simple tools at hand to prove that languages are in one of our context-free-based classes but not in another[1], we are still able to separate the classes with hand-crafted proofs.

5.1. Internal Hierarchy

From Definition 2.7, we can immediately conclude that within the regular- resp. context-free-based classes if the prefix letters of class C are a subset of the prefix letters of class $\mathcal{D}$, then $C \subseteq \mathcal{D}$.

It is also obvious that [F][L][E][D]LLINL $\subseteq$ [F][L][E][D]CFL and [F][L][E][D]RLINL $\subseteq$ [F][L][E][D]CFL. By Corollary 3.7 we also have ωREG $\subseteq \omega$CFL for $\omega \in$ {F,E}. For $\omega \in$ {FE,FED} the result does not follow from the corollary but is still valid.

Lemma 5.5. FEDREG $\subseteq$ FEDCFL.

Proof. Let $L \in$ FEDREG and $G = (N, \Sigma, R, S)$ an FLDRLING generating L. Then L is also generated by the FEDCFG $G' = (N', \Sigma, R', S)$, where $N' = N \sqcup \{A_r \mid r \in R\}$ and

$$R' = \{(A \to A_r \alpha;\ \mathbb{N}^3) \mid r = (A \to \alpha;\ M_r) \in R\} \cup \{(A_r \to \epsilon;\ M_r) \mid r \in R\}. \qquad \square$$

Obviously, the same construction also shows that FEREG $\subseteq$ FECFL. But for FEREG, we can show an even stronger result.

[1] Since the Interchange Lema holds for all classes, it is incapable of separating them.

Lemma 5.6. feREG ⊆ fCFL.

Proof. Let $L \in$ feREG. Then by Theorem 3.9, there is an feDFA $A = (Q, \Sigma, \delta, q_0, F)$ without ϵ-transitions accepting L.

To simulate A using an fPDA B, we will do three things:

1. Simulate A directly, verifying (only) the f-distance on each transition.
2. Simulate A backwards, verifying the e-distance (now the same as the f-distance of B) on each transition.
3. Verify that both simulations use the same sequence of transitions. This way we will verify both distances for each transition, which is sufficient to guarantee the validity of the computation, since A is an feDFA.

It is obvious how to do the forward simulation. For the backward simulation, we will implicitly construct an feNFA for the reversed language, using the construction from Lemma 3.6. Both versions are then run in parallel by making the states of B pairs of states from A, the first component describing the current state of the forward simulation, the second component that of the reverse simulation.

To synchronize the transition sequences we will, during the first half of the computation, record the state sequences of both simulation runs on the stack. In the middle of the computation (which we guess nondeterministically), we switch to a different mode of operation (the states marked with a hat in the formal definition below) in which the simulated runs follow the state sequence recorded by the other run. (Unless this requires a transition that is not allowed at the current position, in which case the simulation stops.)

Since the reverse simulation does not have direct access to the reverse of the input, we have to provide it indirectly. This is done differently for the first and second half of the word.

To provide access to the first half of the word, the forward simulation records it on the stack, along with the sequence of states. During the second half of the computation, the backwards simulation can then read it from there straightforwardly.

For the second half of the word, we resort to nondeterminism: The backwards simulation guesses a possible sequence of characters and records it on the stack. During the second half of the computation, the forward simulation compares this sequence to the actual input.

Since A has no ϵ-transitions, each transition reads a character. This allows us to encode the information put on the stack during each step of the first half of the computation in a single 4-tuple (state of the forward computation, character read by the forward computation, state of the backward computation, character read by the backward computation).

Due to the synchronization of the transitions of both simulation and absence of ϵ-transitions, we are also guaranteed that the current f-distance of B not only matches the f-distance of the forward simulation but also that of the backward simulation (which is the e-distance of A).

Formally, $B = ((Q \times Q) \sqcup (\widehat{Q} \times \widehat{Q}) \sqcup \{q_0', q_f\},\ \Sigma,\ \Gamma = (Q \times \Sigma \times Q \times \Sigma) \sqcup \{g_0\},\ \varphi,\ q_0',\ g_0,\ \{q_f\})$, where $\widehat{Q} = \{\widehat{q} \mid q \in Q\}$ and φ is given by

$$
\begin{aligned}
\varphi(q_0', \epsilon, g_0) &= \{((q_0, q), g_0, \mathbb{N}^3) \mid q \in F\},\\
\varphi((q_1, q_2), a, g) &= \{((q_1', q_2'), g(q_1, a, q_2, b), M \times \mathbb{N}^2) \mid q_1', q_2' \in Q,\ b \in \Sigma,\\
&\qquad M = \{f \in \mathbb{N} \mid \exists e \in \mathbb{N}\colon\ q_1' \in \delta(q_1, a, f, e) \wedge q_2 \in \delta(q_2', b, e-1, f+1)\}\}\\
&\qquad \cup \{((\widehat{q_2}, \widehat{q_1}), g, \{f \in \mathbb{N} \mid q_2 \in \delta(q_1, a, f, f)\} \times \mathbb{N}^2)\}\\
&\qquad \text{for any } q_1, q_2 \in Q,\ a \in \Sigma,\ g \in \Gamma,\\
\varphi((q_1, q_1), \epsilon, g) &= \{((\widehat{q_1}, \widehat{q_1}), g, \mathbb{N}^3)\} \text{ for any } q_1 \in Q,\ g \in \Gamma,\\
\varphi((\widehat{q_1}, \widehat{q_2}), b, (q_2', a, q_1', b)) &= \{((\widehat{q_1}', \widehat{q_2}'), \epsilon, M \times \mathbb{N}^2) \mid\\
&\qquad M = \{f \in \mathbb{N} \mid \exists e \in \mathbb{N}\colon\ q_1' \in \delta(q_1, b, f, e) \wedge q_2 \in \delta(q_2', a, e-1, f+1)\}\}\\
&\qquad \text{for any } q_1, q_2, q_1', q_2' \in Q,\ a, b \in \Sigma,\\
\varphi((\widehat{q}, \widehat{q_0}), \epsilon, g_0) &= \{(q_f, \epsilon, \mathbb{N}^3)\} \text{ for any } q \in F.
\end{aligned}
$$

We now show that for $c \in \Sigma_\epsilon,\ q_1, q_2 \in Q,\ w_1, w_2, u_1, u_2 \in \Sigma^*,\ |u_1| = |u_2|,\ |w_1| = |w_2|,\ \gamma \in \Gamma^*,\ g \in \Gamma$,

$$\begin{aligned} &u_1 q_1 w_1 c w_2 u_2 \vdash_A^{2n+|c|} u_1 w_1 c w_2 q_2 u_2 \\ \rightsquigarrow\ &(u_1(q_1, q_2) w_1 c w_2 u_2, \gamma g) \vdash_B^{4n+1} (u_1 w_1 c w_2(\widehat{q_2}, \widehat{q_1}) u_2, \gamma(|u_1|, M \times \mathbb{N}^2) g) \wedge |u_1| \in M \end{aligned}$$

holds. We do so by induction on n.

Since A has no ϵ-transitions, the computation of A implies that $n = |w_1| = |w_2|$.

$n = 0$: Then $w_1 = w_2 = \epsilon$, reducing the claim to

$$\begin{aligned} &u_1 q_1 c u_2 \vdash_A^{|c|} u_1 c q_2 u_2 \\ \rightsquigarrow\ &(u_1(q_1, q_2) c u_2, \gamma g) \vdash_B^1 (u_1 c(\widehat{q_2}, \widehat{q_1}) u_2, \gamma(|u_1|, M \times \mathbb{N}^2) g) \wedge |u_1| \in M. \end{aligned}$$

We distinguish 2 cases:

$c = \epsilon$: Then the left-hand side further reduces to $u_1 q_1 u_2 \vdash_A^0 u_1 q_2 u_2$, implying $q_1 = q_2$.

By the definition of φ, we also have $((\widehat{q_1}, \widehat{q_1}), g, \mathbb{N}^3) \in \varphi((q_1, q_1), \epsilon, g)$ and thus, $(u_1(q_1, q_1) u_2, \gamma g) \vdash_B (u_1(\widehat{q_1}, \widehat{q_1}) u_2, \gamma(|u_1|, \mathbb{N}^3) g) \wedge |u_1| \in \mathbb{N}$.

The reverse implication follows from the fact that there is no other ϵ-transition applicable to the configuration $(u_1(q_1, q_1) u_2, \gamma g)$.

$c \in \Sigma$: Here, we get on the left-hand side $u_1 q_1 c u_2 \vdash_A u_1 c q_2 u_2$, implying $q_2 \in \delta(q_1, c, |u_1|, |u_2|)$. Thus for $M = \{f \in \mathbb{N} \mid q_2 \in \delta(q_1, c, f, f)\}$, we have $((\widehat{q_2}, \widehat{q_1}), g, M \times \mathbb{N}^2) \in \varphi((q_1, q_2), c, g)$, implying $(u_1(q_1, q_2) c u_2, \gamma g) \vdash_B (u_1 c(\widehat{q_2}, \widehat{q_1}) u_2, \gamma(|u_1|, M \times \mathbb{N}^2) g) \wedge |u_1| \in M$.

Since any other transition in B that processes c either adds another symbol to the stack or removes g, we can again conclude that this is the only possible partial computation on the right-hand side.

$n > 0$: We can partition $w_1 = a w_1',\ w_2 = w_2' b,\ a, b \in \Sigma$ making the computation on the left-hand side

$$u_1 q_1 a w_1' c w_2 u_2 \vdash_A u_1 a q_1' w_1' c w_2' b u_2 \vdash_A^{2n+|c|} u_1 w_1 c w_2' q_2' b u_2 \vdash_A u_1 w_1 c w_2' b q_2 u_2.$$

From the first step, we can conclude that $q_1' \in \delta(q_1, a, |u_1|, |w_1 c w_2 u_2|)$. From the last one, we get $q_2 \in \delta(q_2', b, |u_1 w_1 c w_2'|, |b u_2|)$. Thus, for

$$M = \{f \in \mathbb{N} \mid \exists e \in \mathbb{N}\colon\ q_1' \in \delta(q_1, a, f, e) \wedge q_2 \in \delta(q_2', b, e - 1, f + 1)\}$$

and

$$M' = \{f \in \mathbb{N} \mid \exists e \in \mathbb{N}\colon\ q_1' \in \delta(q_1, a, e - 1, f + 1) \wedge q_2 \in \delta(q_2', b, f, e)\},$$

we have

$$((q_1', q_2'), g(q_1, a, q_2, b), M \times \mathbb{N}^2) \in \varphi((q_1, q_2), a, g)$$

and

$$((\widehat{q_2}', \widehat{q_1}'), \epsilon, M' \times \mathbb{N}^2) \in \varphi((\widehat{q_2}, \widehat{q_1}), b, (q_1', a, q_2', b)).$$

Combining this with the induction hypothesis, we get $\exists M'' \subseteq \mathbb{N}\colon\ |u_1 a| \in M''$ and

$$\begin{aligned} &(u_1(q_1, q_2) a w_1' c w_2' b u_2, \gamma g) \\ \vdash_B\ &(u_1 a(q_1', q_2') w_1' c w_2' b u_2, \gamma(|u_1|, M \times \mathbb{N}^2) g(q_1, a, q_2, b)) \\ \vdash_B^{4n+1}\ &(u_1 w_1 c w_2'(\widehat{q_2}', \widehat{q_1}') b u_2, \gamma(|u_1|, M \times \mathbb{N}^2) g(|u_1 a|, M'' \times \mathbb{N}^2)(q_1, a, q_2, b)) \\ \vdash_B\ &(u_1 w_1 c w_2' b(\widehat{q_2}, \widehat{q_1}) u_2, \gamma(|u_1|, M \times \mathbb{N}^2) g(|u_1 a|, M'' \times \mathbb{N}^2)(|u_1 w_1 c w_2'|, M' \times \mathbb{N}^2)) \\ \vdash_B^2\ &(u_1 w_1 c w_2(\widehat{q_2}, \widehat{q_1}) u_2, \gamma(|u_1|, M \times \mathbb{N}^2) g). \end{aligned}$$

For the inverse implication, we first note that each transition in B that stays in $Q \times Q$ will add one symbol from Γ to the stack, while each transition that stays in $\widehat{Q} \times \widehat{Q}$ will remove one. Since there is no transition from $\widehat{Q} \times \widehat{Q}$ to $Q \times Q$, we can conclude that B stays in $Q \times Q$ while processing w_1 and in $\widehat{Q} \times \widehat{Q}$ while processing w_2. Thus, the symbol put on the stack while reading a must be taken from it while reading b. This is only possible for the symbol (q_1, a, q_2, b). Furthermore, if there were different possibilities than q_1' and q_2' for the components of the intermediate states, this would imply by the definition of φ, resp. the induction hypothesis, a different computation of A on the input, contradicting the fact that A is a DFA.

Thus, we can conclude that there is no other computation for the right-hand side than the one constructed above.

Finally, by the definitions of M and M' we get

$$\begin{aligned}
&\exists e \in \mathbb{N}\colon\ q_1' \in \delta(q_1, a, |u_1|, e),\\
&\exists f \in \mathbb{N}\colon\ q_1' \in \delta(q_1, a, f-1, |u_1 w_1 c w_2'|) = \delta(q_1, a, f-1, |w_1' c w_2 u_2|),\\
&\exists e \in \mathbb{N}\colon\ q_2 \in \delta(q_2', b, |u_1 w_1 c w_2'|, e) \text{ and}\\
&\exists f \in \mathbb{N}\colon\ q_2 \in \delta(q_2', b, f-1, |u_1 a|) = \delta(q_2', b, f-1, |b u_2|).
\end{aligned}$$

Since A is an FEDFA and thus each plset is a Cartesian product of unary sets, these facts imply that $q_1' \in \delta(q_1, a, |u_1|, |w_1 c w_2 u_2|)$ and $q_2 \in \delta(q_2', b, |u_1 w_1 c w_2'|, |b u_2|)$, proving that the computation given at the beginning of the induction step is a valid computation for A.

From the above, we can conclude that for $q \in F$:

$$\begin{aligned}
&q_0 w \vdash_A^* w q\\
\hookrightarrow\ &(q_0', g_0) \vdash_B ((q_0, q)w, (0, \mathbb{N}^3) g_0) \vdash_B^* (w(\widehat{q_0}, \widehat{q}), (0, \mathbb{N}^3)(0, M \times \mathbb{N}^2) g_0) \vdash_B^4 (w q_f, \epsilon),
\end{aligned}$$

thus giving $T(A) = L(B)$. □

From this, FEREG $\subseteq$ ECFL follows immediately with Lemma 2.12 (FEREG is closed under reversal, ECFL is the reversal of FCFL).

The analogous inclusion for LCFL, however, does not hold. We will show this using the language that results from a^* by introducing two different types of marking. One (a' in the formal definition below) marks every character at a f-distance that is a square, the other ($\widehat{a}$ below) does the same for e-distances. Since the two markings are essentially independent, the language can be created by an FLRLING.

Restricting the f- resp. e-distance of a character via length-restrictions directly would require us to derive the complete subword before resp. after that character from a single nonterminal and restrict the rule(s) replacing that nonterminal. However, the words that have to be restricted in our language overlap. As an example, consider the word $a^i \widehat{a} a^j a' a^i,\ i + j + 1 \in \mathbb{N}^{\{2\}}$ (other characters marked appropriately). Here we need to derive each of the subwords $a^i \widehat{a} a^j$ and $a^j a' a^i$ from a single nonterminal, which is impossible in a CFG since they overlap. So the restrictions can not be enforced directly.

To show that there also is no indirect way to achieve this, we will use an interchange argument that makes use of the fact that each sufficiently long subword of a word in our language uniquely determines the word it is a part of and its position in that word.

We will assume an LCFG in CNF for the language and distinguish two cases based on how this grammar generates the majority of words. Either there is a step early in the derivation where both symbols introduced in the step yield a unique (i.e. large) subword, or there is a significant number of steps where a unique subword is split into a short subword and one that is still long enough to be unique.

For the latter case, when looking at all the derivation trees for the language, we will see that there are enough such steps that two of them have to start with the same nonterminal and yield subwords of equal lengths. Interchanging them yields a valid derivation in the grammar but, since the subwords are unique in the language, an invalid word.

For the first case, we have fewer derivation steps and thus have to use a more involved argument. Here, we will show the existence of three derivations of different words w, w' and w'' such that

- they all use the same rule ($A \to BC$) in the step that splits into two large subwords (we denote the subwords that result from B by w_1, w_1', and w_1'' and those that result from C by w_2, w_2', and w_2''), and
- $|w_1'| + |w_2''| = |w_1 w_2|$.

This again allows us to construct a valid derivation of an invalid word:

- We start with the derivation of w.
- The subderivation $B \Rightarrow^* w_1$ is replaced by $B \Rightarrow^* w_1'$.
- The subderivation $C \Rightarrow^* w_2$ is replaced by $C \Rightarrow^* w_2''$.

The new derivation is valid since

- every rule in the subderivation $B \Rightarrow^* w_1'$ is applied with the same length as in the derivation of w',
- every rule in the subderivation $C \Rightarrow^* w_2''$ is applied with the same length as in the derivation of w'', and
- every other rule is applied with the same length as in the derivation of w.

To prove the existence of a triple of words that allows this replacement, we will use a result from graph theory:

Lemma 5.7 (Triangle Removal Lemma, [RS78])**.** *For every $\epsilon > 0$ there is a constant $\delta(\epsilon) > 0$ such that if G is an n-vertex graph that can not be made triangle-free by removing fewer than ϵn^2 edges, then G contains at least $\delta(\epsilon)n^3$ triangles.*

Formalizing the above argument, we get:

Lemma 5.8. *Let*

$$L_2 = \{w \in \{a, a', \widehat{a}, \widehat{a'}\}^* \mid \forall w_1, w_2 \colon\ w = w_1 b' w_2 \rightsquigarrow |w_1| \in \mathbb{N}^{\{2\}},\ b \in \{a, \widehat{a}\};\\ w = w_1 \widehat{b} w_2 \rightsquigarrow |w_2| \in \mathbb{N}^{\{2\}} - \{1\},\ b \in \{a, a'\}\}.$$

$L_2 \in$ feREG $\setminus$ lCFL.

Proof. $L_2 \in$ feREG by the flRLING $G_2 = (\{S\}, \{a, a', \widehat{a}, \widehat{a'}\}, R, S)$ with $M = \mathbb{N}^{\{2\}}$ and

$$\mathrm{R} = \{(S \to aS;\ (\mathbb{N} \setminus M)^2 \times \mathbb{N}))\quad,\ (S \to a'S;\ M \times (\mathbb{N} \setminus M) \times \mathbb{N}),\\ (S \to \widehat{a}S;\ (\mathbb{N} \setminus M) \times M \times \mathbb{N}),\ (S \to \widehat{a'}S;\ M^2 \times \mathbb{N})\qquad,\ (S \to \epsilon;\ \mathbb{N}^3)\}.$$

For any n, the largest difference of two neighbored squares in $[1..n^2]$ is $n^2 - (n-1)^2 = 2n - 1$. Thus for words w with $|w| \leq n^2$, any subword v of w with $|v| \geq 2 \cdot (2n-1) + 2 = 4n \geq 4\sqrt{|w|}$ will contain at least two instances of each of the two types of marking.

Since the distance between two neighboring squares uniquely identifies the squares involved, any such v encodes the number of characters that need to be added to its left and right to extend it to a word in L_2. Since L_2 contains only one word of each length this also means that each such v occurs at only one position in one word in L_2.

Now, assume there is an lCFG $G = (N, \Sigma, R, S)$ that generates L_2. We can assume w.l.o.g. that G is in CNF. For an arbitrary $n > \max\left\{\frac{1}{686|R|} \cdot \delta\left(\frac{1}{98|R|}\right)^{-1},\ (96 \cdot |N|)^2\right\}$, δ the function from the Triangle Removal Lemma, let $S_n = \{w \in L_2 \mid n < |w| \leq 2n\}$ and

$$S_n^{>} = \{w \in S_n \mid S \Rightarrow^* w_0 A w_3 \Rightarrow w_0 B C w_3 \Rightarrow^* w_0 w_1 C w_3 \Rightarrow^* w_0 w_1 w_2 w_3 = w,\\ \min\{|w_1|, |w_2|\} > 4\sqrt{2n}\},$$

that is $S_n^{>}$ contains all the words from S_n where for at least one node in one parse tree both children yield subwords that are unique in L_2. Finally, let $S_n^{<} = S_n \setminus S_n^{>}$.

Since $|S_n| = n$, we have either $|S_n^{>}| \geq \frac{n}{2}$ or $|S_n^{<}| > \frac{n}{2}$. We will treat these cases separately.

$|S_n^>| \geq \frac{n}{2}$: There is at least one rule $A \to BC$ that appears in a partitioning according to the definition of $S_n^>$ of the derivations of at least $\frac{n}{2|R|}$ words from $S_n^>$. We will call S_n' the corresponding subset of $S_n^>$. For words that have multiple such partitionings, fix an arbitrary one.

Assume the existence of w and w' two different words from S_n' so that the subwords from the partitioning satisfy $|w_1| = |w_1'|$. Since by definition of $S_n^>$, they are long enough to be unique in L_2, we can conclude that $w_1 \neq w_1'$. Thus $|w_0 w_1' w_2 w_3| = |w|$ but $w_0 w_1' w_2 w_3 \neq w$. Since L_2 contains only one word of each length, we can conclude that $w_0 w_1' w_2 w_3 \notin L_2$. However,

$$S \Rightarrow^* w_0 A w_3 \Rightarrow w_0 B C w_3 \Rightarrow^* w_0 w_1' C w_3 \Rightarrow^* w_0 w_1' w_2 w_3$$

is a valid derivation in the LCFG G, since each rule in the first and last part is applied with the same length as in the derivation of w, and each rule in between is applied with the same length as in the derivation of w'. Thus, we may assume that $|w_1|$ (and by analogous argument $|w_2|$ and $|w_1 w_2|$) is different for each word in S_n'. $(*)$

We will now use the Triangle Removal Lemma to show the existence of pairwise distinct w, w', and w'' so that $|w_1'| + |w_2''| = |w_1 w_2|$. This makes

$$S \Rightarrow^* w_0 A w_3 \Rightarrow w_0 B C w_3 \Rightarrow^* w_0 w_1' C w_3 \Rightarrow^* w_0 w_1' w_2'' w_3$$

a valid derivation in G, since each rule in the first part is applied with the same length as in the derivation of w, each rule in the second part is applied with the same length as in the derivation of w', and each rule in the third part is applied with the same length as in the derivation of w''.

Since w_1' and w_2'' appear in L_2 only as subwords of w' resp. w'' and $w' \neq w''$, we can conclude that $w_0 w_1' w_2'' w_3 \in L(G) \setminus L_2$.

The idea for the application of the Triangle Removal Lemma is to create a graph with the following properties:

(1) Each edge represents one subword w_1, w_2, or $w_1 w_2$ from one of the words in S_n'. Each of the subwords is represented by n of the edges.

(2) Three edges form a triangle only if they represent subwords w_1', w_2'' and $w_1 w_2$ (w, w' and w'' not necessarily distinct) so that $|w_1'| + |w_2''| = |w_1 w_2|$.

(3) For each word in S_n', the edges representing its subwords form n node-disjoint triangles, giving a total of $\Theta(n^2)$ triangles.

Since the graph will have $\Theta(n)$ nodes we can conclude from (3) with the Triangle Removal Lemma that if n is sufficiently large, there must be additional triangles. Since the edges corresponding to the subwords of each word from S_n' form disjoint triangles, the additional triangles must come from edges representing subwords of pairwise distinct words. With this, (2) gives the desired property.

Formally, the graph contains three groups of vertices

$$A = \{v_{a,i}, \mid 0 \leq i < n\},\ B = \{v_{b,i}, \mid 0 \leq i < 3n\},\ \text{and } C = \{v_{c,i}, \mid 0 \leq i < 3n\}.$$

For each $w \in S_n'$ and each i, $0 \leq i < n$ we add a triangle consisting of the edges

$$(v_{a,i}, v_{b,i+|w_1|}),\ (v_{b,i+|w_1|}, v_{c,i+|w_1 w_2|}) \text{ and } (v_{c,i+|w_1 w_2|}, v_{a,i}).$$

For each edge, the first indices of its endpoints identify the type of subword it corresponds to: The first group of edges corresponds to subwords w_1, the second group corresponds to subwords w_2, and the third group corresponds to subwords $w_1 w_2$, thus no edge can correspond to multiple subwords of the same word. The difference between the two second indices gives the length of the encoded subword. Since by $(*)$, these lengths are different for each word in S_n', no edge can correspond to subwords from two different words. Together with the trivial observation that n distinct edges representing each subword are explicitly added, property (1) is proven.

By the definition of the edges, each triangle in the graph must consist of three nodes v_{a,i_1}, v_{b,i_2}, and v_{c,i_3}. By the observations in the previous paragraph, the edges of such a triangle represent subwords w_1', w_2'', and w_1w_2 such that

$$\begin{aligned} |w_1'| &= i_2 - i_1, \\ |w_2''| &= i_3 - i_2 \text{ and} \\ |w_1w_2| &= i_3 - i_1 = |w_1'| + |w_2''|. \end{aligned}$$

This shows property (2).

Concerning property (3), it is again straightforward from the definition of the graph that the edges representing the subwords of each word form n node-disjoint (and thus also edge-disjoint) triangles. Thus to make the graph triangle free, at least one edge from each of these $n \cdot \frac{n}{2|R|} = \frac{1}{98|R|}(7n)^2$ triangles has to be removed from the graph with $7n$ nodes.

Thus, by the Triangle Removal Lemma the graph contains at least $\delta\left(\frac{1}{98|R|}\right)(7n)^3$ triangles. For $n > \frac{1}{686|R|} \cdot \delta\left(\frac{1}{98|R|}\right)^{-1}$, this means that there is a triangle that we did not explicitly add, giving us a word in $L(G) \setminus L_2$.

$|S_n^<| > \frac{n}{2}$: Consider a parse tree of a word $w \in S_n^<$. By starting at the root node and descending into the child corresponding to the larger subword $\frac{\sqrt{n}}{16}$ times we visit $\frac{\sqrt{n}}{16}$ nodes.

By definition of $S_n^<$, the child we do not visit in a step corresponds to a subword of length at most $4\sqrt{2n}$. Thus, the $(i+1)$-st node we visit corresponds to a subword with a length of at least

$$n - i \cdot 4\sqrt{2n} > n - i \cdot 8\sqrt{n} \geq n - \frac{\sqrt{n}}{16} \cdot 4\sqrt{2n} = \frac{1}{2}n.$$

Thus in the parse trees of all words in $S_n^<$ combined we find at least $\frac{n}{4} \cdot \frac{1}{16}\sqrt{n} = \frac{1}{64}n\sqrt{n}$ nodes corresponding to subwords of sizes between $\frac{1}{2}n$ and $2n$. So one of the lengths has to appear at least $\frac{2}{3n}\frac{1}{64}n\sqrt{n} = \frac{1}{96}\sqrt{n}$ times. For $n > (96 \cdot |N|)^2$, this means at least two of these words will be derived from the same nonterminal. Exchanging the corresponding partial derivations as in the previous case leads to a derivation in G of a word not in L_2.

Since in both cases $L(G) \neq L_2$, the lemma is proven. □

This result might lead to the idea that length-dependency is weaker than position-dependency and one might question if the former can be simulated by the latter. This is not the case: Looking at the language $a^*b^{n^2}a^*$ we note that the restricted subword can start and end at arbitrary positions. Thus, position restrictions do not appear to be helpful, while the language can trivially be generated using length restrictions.

A formal proof that this language can not be generated by an fedCFG is hindered by the fact that a context-free grammar can generate words from $a^*b^*a^*$ in many different ways. This makes it difficult to get subderivations that can be interchanged as in the previous lemma.

To work around this problem, we replace the three parts by Dyck words (over square brackets), separated by markers, making the language L_3 contain the words $w_1aw_2aw_3$, with Dyck words w_i, and the additional constraint $|w_2| \in \{2\}^{\mathbb{N}}$. By a straightforward application of Theorem 5.3, we will show that the Dyck language is not in fedREG. This establishes that correct parenthesization can not be ensured by position (or length) restrictions and thus has to be ensured by the underlying context-free grammar. This guarantees that a hypothetical grammar for L_3 has some nice properties:

- For each nonterminal A, the degree of unbalancedness, i.e. the value $|v|_[- |v|_]$, is the same for all words $v \in \Sigma^*$ that can be derived from A. (If $A \Rightarrow^* v$ and $A \Rightarrow^* v'$, we may replace instances of v that have been derived from A by v'. If $|v|_[- |v|_] \neq |v'|_[- |v'|_]$, this would result in a word that is not in the language.)

- There is a constant k so that every derivation of $w = w'[^m]^m w''$, $m > k$, can be partitioned as $S \Rightarrow^* w'[^{i_1}A]^{i_2}w'' \Rightarrow^* w$, with $i_1, i_2 \leq k$. (If k' denotes the maximal degree of unbalancedness for any nonterminal in the grammar, we can introduce at most k' closing parentheses before introducing the first of the opening ones. If k'' denotes the maximal number of pairs of parentheses introduced by any rule of the grammar, we get $k' + k''$ as a bound for i_2. Analogously, we can derive a bound for i_1 and set k to the maximum of those bounds.)

Now, we look at the sublanguage of L_3, where each w_i is of the form $[^n]^n$. By the above considerations, we find that for $w = [^o]^o a[^n]^n a[^p]^p \in L_3$ and $m < n$, each derivation of w can be partitioned as

$$\begin{aligned} S &\Rightarrow^q [^{i_1}A_1]^{i_2} a[^{i_3}A_2]^{i_4} a[^{i_5}A_3]^{i_6} \\ &\Rightarrow^* [^o]^o a[^{i_3}A_2]^{i_4} a[^{i_5}A_3]^{i_6} \\ &\Rightarrow^* [^o]^o a[^{i_3}A_2]^{i_4} a[^p]^p \\ &\Rightarrow^* [^o]^o a[^{n-m}[^{i_7}A_4]^{i_8}]^{n-m} a[^p]^p \\ &\Rightarrow^* [^o]^o a[^{n-m}[^m]^m]^{n-m} a[^p]^p, \end{aligned} \tag{5.2}$$

where $i_j \leq k$, $1 \leq j \leq 8$ and, assuming a grammar in CNF, $q = 6 + \sum_{j=1}^{6} 2i_j \leq 12k + 6$.

Now, we can represent these partitioned derivations by 4-tuples (m, n, o, p) and put these into equivalence classes based on the first part of the derivation (which also fixes $i_1, \ldots, i_6$ and $A_1, \ldots, A_3$), i_7, i_8, and A_4. (For ambiguous words, we can simply choose any of their derivations, even a different one for each value of m.) There are at most $c = |R|^q + 2k + |N|$ different equivalence classes.

If we now find a set of equivalent tuples (m, n, o, p), $(m + 2\lambda, n, o + \lambda, p + \lambda)$, $(m, n, o + 2\lambda, p)$ and $(m, n, o, p + 2\lambda)$, we can recombine parts of their corresponding derivations into a new derivation:

$$\begin{aligned} S &\Rightarrow^q [^{i_1}A_1]^{i_2} a[^{i_3}A_2]^{i_4} a[^{i_5}A_3]^{i_6} \\ &\Rightarrow^* [^o]^o a[^{i_3}A_2]^{i_4} a[^{i_5}A_3]^{i_6} \\ &\Rightarrow^* [^o]^o a[^{i_3}A_2]^{i_4} a[^p]^p \\ &\Rightarrow^* [^o]^o a[^{n-m}[^{i_7}A_4]^{i_8}]^{n-m} a[^p]^p \\ &\Rightarrow^* [^o]^o a[^{n-m}[^{m+2\lambda}]^{m+2\lambda}]^{n-m} a[^p]^p = w'. \end{aligned}$$

Since the tuples are equivalent, the values of $i_1, \ldots, i_8$ and $A_1, \ldots, A_4$ are the same for each of the original derivations, allowing the recombination in the underlying CFG. To verify that this derivation is valid with respect to the plsets, we look at the parts separately:

- The partial derivation
$$A_1 \Rightarrow[(i_1,\ 2o-i_1-i_2,\ e)]\Rightarrow^* [^{o-i_1}]^{o-i_2}$$
appears with the same position as in the derivation corresponding to $(m, n, o, p + 2\lambda)$, since
$$e \ = \ i_2 + 1 + 2(n + 2\lambda) + 1 + 2p \ = \ i_2 + 1 + 2n + 1 + 2(p + 2\lambda).$$

- The partial derivation
$$A_3 \Rightarrow[(f,\ 2p-i_5-i_6,\ i_6)]\Rightarrow^* [^{p-i_5}]^{p-i_6}$$
appears with the same position as in the derivation corresponding to $(m, n, o + 2\lambda, p)$, since
$$f \ = \ 2o + 1 + 2(n + 2\lambda) + 1 + i_5 \ = \ 2(o + 2\lambda) + 1 + 2n + 1 + i_5.$$

- The partial derivation
$$A_2 \Rightarrow[(2o + 1 + i_3,\ l,\ i_4 + 1 + 2p)]\Rightarrow^* [^{n-m-i_3}[^{i_7}A_4]^{i_8}]^{n-m-i_4}$$
appears with the same position as in the derivation corresponding to (l, m, n, o). (l differs for the two contexts. Since this is only due to the different yields of A_4 and the grammar is not length-dependent, it does not influence this part of the derivation.)

- The partial derivation

$$A_4 \Rightarrow\!\!\left[(f,\ 2(m+2\lambda)-i_7-i_8,\ e)\right]\!\!\Rightarrow^* [^{m+2\lambda-i_7}]^{m+2\lambda-i_8}$$

  appears with the same position as in the derivation corresponding to $(m+2\lambda, n, o+\lambda, p+\lambda)$, since

$$f \ = \ 2o+1+n-m+i_7 \ = \ 2(o+\lambda)+1+n-(m+2\lambda)$$

  and

$$e \ = \ i_8+n-m+1+2p \ = \ i_8+n-(m+2\lambda)+1+2(p+\lambda).$$

- For the partial derivation

$$S \Rightarrow^q [^{i_1}A_1]^{i_2}a[^{i_3}A_2]^{i_4}a[^{i_5}A_3]^{i_6},$$

  we have to treat the individual rules separately. We distinguish three cases based on the yield of the rules:
  - If the yield includes A_2, the rule appears with the same position as in the derivation corresponding to (m, n, o, p).
  - If the yield is left of A_2, the rule appears with the same position as in the derivation corresponding to $(m, n, o, p+\lambda)$.
  - If the yield is right of A_2, the rule appears with the same position as in the derivation corresponding to $(m, n, o+\lambda, p)$.

By the definition of L_3, n is a power of 2. If $2\lambda < n$ then $n+2\lambda < 2n$ is not a power of 2 and thus $w' = [^o]^o a[^{n+2\lambda}]^{n+2\lambda} a[^p]^p \notin L_3$, but it is generated by the grammar.

So in order to complete our proof, we need to show that four equivalent tuples as described above exist for each fedCFG. To do so, we use a result from Ramsey theory:

Theorem 5.9 (Gallai's Theorem, [GRS80])**.** *For every $\mathcal{S} \subset \mathbb{N}^d$, $c \in \mathbb{N}$, there is $N = N(\mathcal{S}, c)$ so that for any c-coloring of $\{1, \dots, N\}^d$ there is $a \in \mathbb{N}^d$, $\lambda \in \mathbb{N}_+$, so that $\{a\} + \{\lambda\} \cdot \mathcal{S}$ is monochromatic.*

In our setting, the colors correspond to the equivalence classes. In order to ensure $2\lambda < n$, we will not use the 4-tuples defined above directly. Instead, we define for each $n = 2^{n'}$, $n' \in \mathbb{N}$, the set

$$S_n = \{(m, o, p) \mid 1 \leq m, o, p \leq n-k\},$$

where the tuple $(m, o, p) \in S_n$ corresponds to the tuple $(m+k, n, o+k, p+k)$ above.

For each set S_n, we define a c-coloring of $S_n = \{1, \dots, n-k\}^3$ by assigning a color to each equivalence class of the tuples and coloring each tuple according to its equivalence class. By setting

$$\mathcal{S} = \{(0,0,0),\ (2,1,1),\ (0,2,0),\ (0,0,2)\},$$

and choosing $n > N(\mathcal{S}, c) + k$, we get by Gallai's Theorem the existence of $a = (m', o', p')$ and $\lambda \in \mathbb{N}$ so that $\{a\} + \{\lambda\} \cdot \mathcal{S}$ is monochromatic in the coloring implied by S_n. This implies that for $m = m' + k$, $o = o' + k$, and $p = p' + k$ the original tuples

$$\{(m, n, o, p)\} + \{\lambda\} \cdot \mathcal{S} = \{(m, n, o, p),\ (m+2\lambda, n, o+\lambda, p+\lambda),\ (m, n, o+2\lambda, p),\ (m, n, o, p+2\lambda)\}$$

are equivalent as desired.

We also find by the definition of S_n:

$$2\lambda < 2\lambda + m' \leq n - k < n.$$

Thus $w' \notin L_3$, as explained above.

We complete our proof with the promised lemma that the Dyck language is not in fedREG:

Lemma 5.10. *Let* $L_D = \{w \in \{[,]\}^* \mid w \text{ is correctly parenthesized}\}$. $L_D \in \text{CFL} \setminus \text{FEDREG}$.

Proof. It is well-known that $L_D \in \text{CFL}$.

Now for $n, m \in \mathbb{N},\ 1 \leq m < \frac{n}{2}$, consider the words $w_{n,m} = [^{n-m}]^m$ and $w'_{n,m} = [^m]^{n-m}$. We have $w_{n,m_1} w'_{n,m_2} \in L_D \rightarrowtail m_1 = m_2$. Thus $R_{L_D,(n,n)}$ induces a separate equivalence class for each possible value of m. Since the number of values for m increases with n, the number of equivalence classes can not be bounded by a constant. Theorem 5.3 thus yields $L_D \notin \text{FEDREG}$. □

Summing up the result of the last two pages, we get:

Lemma 5.11. *Let* $L_3 = \{w_1 a w_2 a w_3 \mid w_1, w_2, w_3 \in L_D, |w_2| \in \{2\}^{\mathbb{N}}\}$. $L_3 \in \text{LCFL} \setminus \text{FEDCFL}$.

Proof. $L_3 \in \text{LCFL}$ by the LCFG $G_3 = (\{S, T, D\}, \{[,], a\}, R, S)$, with

$$R = \{(S \to DaTaD;\ \mathbb{N}^3),\ (T \to D;\ \mathbb{N} \times \{2\}^{\mathbb{N}} \times \mathbb{N}),\ (D \to [D]D;\ \mathbb{N}^3),\ (D \to \epsilon;\ \mathbb{N}^3)\}.$$

□

In a similar fashion, we can show that one type of position restrictions together with length restrictions can not replace the other type of position restrictions.

Lemma 5.12. *Let* $L_4 = \{w_1 a^o w'_1 a w_2 \mid w_1 w'_1, w_2 \in L_D,\ |w_1| \in \{2\}^{\mathbb{N}}, o \in \mathbb{N}\}$. $L_4 \in \text{FCFL} \setminus \text{LEDCFL}$.

Proof. $L_4 \in \text{FCFL}$ by the FCFG $G_4 = (\{S, A, B, D\}, \{[,], a\}, R, S)$, with

$$\begin{aligned} R = \{&(S \to BaD;\ \mathbb{N}^3),\ (B \to [B]D;\ \mathbb{N}^3),\ (A \to aA;\ \mathbb{N}^3)\ ,\ (A \to \epsilon;\ \mathbb{N}^3),\\ &(B \to A;\ \{2\}^{\mathbb{N}} \times \mathbb{N}^2)\ ,\ (B \to [D]B;\ \mathbb{N}^3),\ (D \to [D]D;\ \mathbb{N}^3),\ (D \to \epsilon;\ \mathbb{N}^3)\}. \end{aligned}$$

Now, assume there is an LEDCFG G in CNF that generates L_4. As in the previous lemma, the balancedness of the parentheses can not be ensured by position or length restrictions but has to be ensured by the underlying context-free grammar. Thus, just as in Equation (5.2), we find that for each $w = [^n a^o]^n a [^p]^p \in L_4$ and $m < n$, each derivation of w can be partitioned as

$$\begin{aligned} S &\Rightarrow^q [^{i_1} A_1]^{i_2} a [^{i_3} A_2]^{i_4} \\ &\Rightarrow^* [^{i_1} A_1]^{i_2} a [^p]^p \\ &\Rightarrow^* [^m [^{i_5} A_3]^{i_6}]^m a [^p]^p \\ &\Rightarrow^* [^m [^{n-m} a^o]^{n-m}]^m a [^p]^p, \end{aligned}$$

where $i_j \leq k$, $1 \leq j \leq 6$ and, since G is in CNF, $q = 3 + \sum_{j=1}^{4} 2i_j \leq 8k + 3$.

As in the previous proof, we can identify each partitioned derivation with the tuple (m, n, o, p) and group the tuples into equivalence classes based on the first part of the derivation, i_5, i_6, and A_3. Again, the number of equivalence classes is bound by $c = |R|^q + 2k + |N|$.

In order to apply Gallai's Theorem, we again define for each $n = 2^{n'}$, $n' \in \mathbb{N}$, the set

$$S_n = \{(m, o, p) \mid 1 \leq m, o, p \leq n - k\},$$

where the tuple $(m, o, p) \in S_n$ corresponds to the tuple $(m + k, n, o + k, p + k)$ above. Now for

$$\mathcal{S} = \{(0, 0, 0),\ (0, 0, 1),\ (2, 4, 0)\}$$

and $n > N(\mathcal{S}, c) + k$, we get by Gallai's Theorem the existence of $a = (m', o', p')$ and $\lambda \in \mathbb{N}$ so that $\{a\} + \{\lambda\} \cdot \mathcal{S}$ is monochromatic in the coloring implied by S_n. This implies that for $m = m' + k$, $o = o' + k$ and $p = p' + k$ the original tuples

$$\{(m, n, o, p)\} + \{\lambda\} \cdot \mathcal{S} = \{(m, n, o, p),\ (m, n, o, p + \lambda),\ (m + 2\lambda, n, o + 4\lambda, p)\}$$

are equivalent, allowing us to recombine parts of the corresponding derivations to a new derivation:

$$\begin{aligned} S &\Rightarrow^q [^{i_1}A_1]^{i_2}a[^{i_3}A_2]^{i_4} \\ &\Rightarrow^* [^{i_1}A_1]^{i_2}a[^{p}]^{p} \\ &\Rightarrow^* [^{m+2\lambda}[^{i_5}A_3]^{i_6}]^{m+2\lambda}a[^{p}]^{p} \\ &\Rightarrow^* [^{m+2\lambda}[^{n-m}a^{o}]^{n-m}]^{m+2\lambda}a[^{p}]^{p} = w'. \end{aligned}$$

Since the tuples are equivalent, the values of $i_1, \dots, i_6$ and $A_1, \dots, A_3$ are the same for each of the original derivations, allowing the recombination in the underlying CFG. To verify that this derivation is valid with respect to the plsets, we look at the parts separately:

- The partial derivation
$$A_2 \Rightarrow\!\!\!|(f,\ 2p-i_3-i_4,\ i_4)|\!\!\Rightarrow^* [^{p-i_3}]^{p-i_4}$$
appears with the same e-distance and length as in the derivation corresponding to (m, n, o, p). (f differs for the two contexts. This is not a problem since G is a LEDCFG.)

- The partial derivation
$$A_1 \Rightarrow\!\!\!|(i_1,\ l,\ i_2 + 1 + 2p)|\!\!\Rightarrow^* [^{m+2\lambda-i_1}[^{i_5}A_3]^{i_6}]^{m+2\lambda-i_2}$$
appears with the same e-distance and length as in the derivation corresponding to the tuple $(m+2\lambda, n, o+4\lambda, p)$, since
$$l \;=\; 2(n+2\lambda) + o - i_1 - i_2 \;=\; 2n + (o+4\lambda) - i_1 - i_2.$$

- The partial derivation
$$A_3 \Rightarrow\!\!\!|(f,\ 2(n-m)+o,\ e)|\!\!\Rightarrow^* [^{n-m-i_5}a^{o}]^{n-m-i_6}$$
appears with the same e-distance and length as in the derivation corresponding to $(m, n, o, p+\lambda)$, since
$$e \;=\; i_6 + (m+2\lambda) + 1 + 2p \;=\; i_6 + m + 1 + 2(p+\lambda).$$

- For the partial derivation
$$S \Rightarrow^q [^{i_1}A_1]^{i_2}a[^{i_3}A_2]^{i_4}a[^{i_5}A_3]^{i_6}$$
we have to treat the individual rules separately. We distinguish two cases based on their yield:
  - If the yield includes A_1 or is left of A_1, the rule appears with the same e-distance and length as in the derivation corresponding to $(m+2\lambda, n, o+4\lambda, p)$.
  - If the yield is right of A_1, the rule appears with the same e-distance and length as in the derivation corresponding to (m, n, o, p).

Since furthermore $2\lambda < 2\lambda + m' \leq n - k < n$ holds, we can conclude that $w' \in L(G) \setminus L_4$. □

From this, we get $L_4^R \in$ ECFL $\setminus$ FLDCFL by a straightforward application of Lemma 2.12.

For the regular-based language classes we get the equivalent result: One type of position restrictions does not suffice to emulate the other one. The proof is considerably easier, though.

Lemma 5.13. *Let* $L_5 = \{b^m a^{n^2} \mid m, n \in \mathbb{N}\}$. $L_5 \in$ EREG $\setminus$ (FREG $\cup$ CFL).

Proof. L_5 is generated by the LRLING $G_5 = (\{A, S\}, \{a, b\}, R, S)$, with

$$R = \{(S \to bS;\ \mathbb{N}^3),\ (S \to A;\ \mathbb{N} \times \mathbb{N}^{\{2\}} \times \mathbb{N}),\ (A \to aA;\ \mathbb{N}^3),\ (A \to \epsilon;\ \mathbb{N}^3)\}.$$

Now for $n \in \mathbb{N}$, consider $\{w_{n,m} = b^m a^{n^2-m} \mid 0 < m < 2n-1\}$. Then for $0 < k < 2n-1$, we have $w_{n,m}a^k \in L_5 \leftrightarrows k = m$. Thus, $R_{L_5,(f,*)}$ induces a separate equivalence class for each possible value of m. Since the number of those is not bounded by a constant, Theorem 5.3 yields $L_5 \notin$ FREG.

It is easily verified that $L_5 \notin$ CFL, using e.g. the pumping Lemma for context-free languages. □

This leaves us with only one open question concerning the internal hierarchy: Do dependencies of restrictions add additional power or not?

On an intuitive level, this question is easily answered by noting that dependent plsets can encode relative positions or lengths as seen in the grammar G_1 in Example 2.5 that generates $a^n b^n c^n$ essentially by requiring the a's and c's to make up a third of the length of the word each. Plsets that are the Cartesian Product of unary sets cannot express such relations.

This does, however, not exclude the possibility that the relations are captured through interactions between different plsets and the underlying CFG. As an example, consider the FCFG $G_6 = (\{S, B, E\}, \{a, b, c\}, R, S))$, with

$$R = \{(S \to aSc;\ \mathbb{N}^3)\ ,\quad (B \to bBb;\ \mathbb{N}^3)\quad ,\\ (S \to EBE;\ \mathbb{N}^3),\ r_1 = (B \to \epsilon;\ \{2\}^{\mathbb{N}} \cdot \{3\} \times \mathbb{N}^2),\ r_2 = (E \to \epsilon;\ \{2\}^{\mathbb{N}} \times \mathbb{N}^2)\}.$$

Rule r_2 ensures that the number of a's is 2^{n_1} and the number of a's and b's combined is 2^{n_3}. Rule r_1 guarantees that the number of a's plus half the number of b's is $3 \cdot 2^{n_2}$. Thus, we get $2^{n_3} - 3 \cdot 2^{n_2} = 3 \cdot 2^{n_2} - 2^{n_1}$ or equivalently

$$2^{n_3} + 2^{n_1} = 3 \cdot 2^{n_2+1}.$$

To extract information about n_1, n_2, and n_3 from this equation, we look at the binary representation of the numbers. For the number on the right-hand side of the equation, the binary representation is 110^{n_2+1}. For the number on the left-hand side, we distinguish two cases:

$n_1 = n_3$: This gives the binary representation 10^{n_1+1}, which does not match that on the right-hand side.

$n_1 \neq n_3$: From the structure of the grammar, we know that $n_3 \leq n_1$. Thus, we get the binary representation $10^{n_3-n_1-1}10^{n_1}$. This matches the right-hand side if and only if $n_3 - n_1 - 1 = 0$ and $n_1 = n_2 + 1$, or equivalently, $n_3 = n_1 + 1 = n_2 + 2$.

From this, we can conclude that G_6 generates $\{a^n b^n c^n \mid n \in \{2\}^{\mathbb{N}_+}\}$, something that our initial intuition would have deemed impossible for an FLECFG.

Additionally, the proof technique from Lemmas 5.11 and 5.12 does not work here. If we could combine parts of different valid derivations into a derivation of a word that is not in the language, the same construction would also work for an FLEDCFG. But our goal is to separate FLECFL and FLEDCFL.

We could solve this problem by looking for a partial derivation that appears with positions (f_1, e_1) as well as (f_2, e_2) in valid derivations, and that can be used with positions (f_1, e_2) in an invalid derivation. However, this introduces a new problem: Since the grammar may be ambiguous, it is not sufficient to identify the partial derivation by the nonterminal it starts with and the subword that results from it. In the terms of the proofs of Lemmas 5.11 and 5.12, we need to encode the specific derivation into the equivalence class. Since the number of equivalence classes has to be bounded by a constant, the same now holds for the size of the subderivation that appears multiple times. Unfortunately, this prohibits the application of Gallai's Theorem in the style used above, since that would require the size of the partial derivation to be unbounded.

Since all our attempts to prove that dependent plsets are more powerful were equally unsuccessful, we have to resort to stating this relation as a conjecture.

Conjecture 5.14. FEDREG \ FLECFL $\neq \emptyset$.

While none of our attempts have yielded a formal proof of the statement, some of them have at least given strong arguments in its support:

- Taking $a^n b^n c^n$ as a candidate language for being in the difference, we note that by Lemma 5.16 from the following section, it can not be generated by an FLECFG where each plset is semilinear since that would imply that the language is context-free.

  We can furthermore conclude that the language can not be generated by an FLECFG where the underlying CFG is unambiguous:

  Assume to the contrary that $G = (N, \Sigma, R, S)$ is an unambiguous FLECFG in CNF that generates $a^n b^n c^n$. In a first step, we will turn G into an unambiguous FLECFG $G' = (N',\ \{a, b, c\},\ R',\ S')$ so that the underlying CFG of G' generates a subset of $a^* b^* c^*$:

  We set $N' = \{A_\Phi \mid A \in N,\ \Phi \subseteq \{a, b, c\}\},\ S' = S_{\{a,b,c\}}$ and

  $$R' = \{(A_\Phi \to B_{\Phi'} C_{\Phi''};\ M) \mid (A \to BC;\ M) \in R,\ \mathrm{cond}(\Phi, \Phi', \Phi'')\}$$
  $$\cup\ \{(A_{\{x\}} \to x;\ M) \mid (A \to x;\ M) \in R,\ x \in \{a, b, c\}\},$$

  where $\mathrm{cond}(\Phi, \Phi', \Phi'')$ describes the following conditions:

  - $\Phi' \cup \Phi'' = \Phi$.
  - Φ'' only contains letters that may appear right of each of the letters in Φ' and vice versa. (So if e.g., $c \in \Phi'$, then $\Phi'' = \{c\}$; if $b \in \Phi''$, then $\Phi' \subseteq \{a, b\}$.)

  It is easily verified that the underlying CFG of G' only generates a subset of $a^* b^* c^*$. To see that $L(G') = L(G)$, consider any valid parse tree in G and replace each instance of a nonterminal A by A_Φ, where Φ describes the set of letters in the yield of the instance in question. This modified tree is a valid parse tree in G':

  - The position and length of any rule application in the annotated tree is identical to that of the corresponding rule application in the original tree.
  - The yield of any nonterminal is the union of the yields of its children.
  - The yield of a left (resp. right) child will never contain letters that may appear right (left) of the letters in the yield of its sibling.

  This shows $L(G') \supseteq L(G)$. Since removing the annotations from all nonterminals turns any valid parse tree in G' into one in G, we also have $L(G') \subseteq L(G)$ as desired.

  Now we will show that we can generate $a^n b^n c^n$ using the underlying CFG of G' and only semilinear plsets. Together with Lemma 5.16, this proves our claim. The basic idea behind our construction is to define the plsets so they contain exactly those f-distances, lengths and e-distances at which the respective rule appears in the derivation of a word in $a^n b^n c^n$. Since the underlying CFG is unambiguous, each of these rule applications is necessary in order to generate the language. (We can conclude that the plsets of G' are a superset of these.) On the other hand, these sets are already sufficient to generate the language. Thus, by showing that these are semilinear, we complete our proof:

  For each rule r in G' consider the sets $M_{r,f}$, $M_{r,l}$ and $M_{r,e}$ consisting of all 4-tuples (f, n_a, n_b, n_c), (l, n_a, n_b, n_c), resp. (e, n_a, n_b, n_c), where r appears in the derivation of $a^{n_a} b^{n_b} c^{n_c}$ with f-distance f, length l, resp. e-distance e. As G' is a CFG, each of these sets is semilinear.

  Since semilinear sets are closed under intersection and projection the sets $M'_{r,f} = \Pi_1(M_{r,f} \cap M)$, $M'_{r,l} = \Pi_1(M_{r,l} \cap M)$, and $M'_{r,e} = \Pi_1(M_{r,e} \cap M)$ are also semilinear, where Π_1 denotes projection to the first component and $M = \{(m, n, n, n) \mid m, n \in \mathbb{N}\}$. Since $M'_{r,f} \times M'_{r,l} \times M'_{r,e}$ are exactly the plsets described above, we are done.

- There is no construction that turns each FLDRLING into an equivalent FLECFG such that the structure of the FLECFG, i.e. the underlying CFG, only depends on the structure of the FLDRLING. We will show this by a counting argument:

  There is an obvious bijection between sets $M \subseteq \mathbb{N}^2$ and languages $L \subseteq a^* b^*$, where M corresponds to $L_M = \{a^p b^q \mid (p, q) \in M\}$. Also, $L_M \in$ FEDREG by the FLDRLING $G_M = (\{S, A\}, \{a, b\}, R_M, S)$ with

  $$R_M = \{(S \to aS;\ \mathbb{N}^3),\ (S \to A;\ M),\ (A \to bA;\ \mathbb{N}^3),\ (A \to \epsilon;\ \mathbb{N}^3)\}.$$

Now, for any $n \in \mathbb{N}$ define $\mathcal{L}^{(n)} = \{L_M \mid M \subseteq \{1, \dots, n\}^2\}$ the set of languages L_M that only contain words of length at most n. Since there are $\sum_{i=0}^{n}(i+1) = \frac{n^2+3n+2}{2}$ such words over a^*b^*, there are $2^{\frac{n^2+3n+2}{2}}$ languages in $\mathcal{L}^{(n)}$.

Since the grammars G_M all have the same structure (they only differ in a single plset), a construction satisfying the criterion from above would turn them into a set of FLECFG that only differs on the plsets on their rules. Denote by c the number of rules of these grammars.

Since the derivation of a word of length at most n will only depend on entries (f, l, e) of the plsets where each component is at most n, the languages in $\mathcal{L}^{(n)}$ are already recognized by grammars in which no plset contains an entry with a component larger than n. Since there are 2^{3n} different possibilities for each plset on these entries, the number of languages in $\mathcal{L}^{(n)}$ recognized by such grammars is 2^{3cn}. For $n > 6c$, this is less than $|\mathcal{L}^{(n)}|$.

These observations correspond nicely with the way we treated plsets that were finite unions of FLE-plsets in the proofs of Lemma 2.14 and Theorem 3.9: We introduced multiple rules resp. transitions that only differed in their plsets. On the underlying CFG resp. automaton, this has the effect of introducing additional ambiguities.

Summing up the results of this section, we get:

Theorem 5.15. *The inclusion hierarchy of the language classes considered in this thesis is as depicted in Figure 5.2. There is an arrow or path from Class C to class C′ if and only if $C \subsetneq C'$. Thinner blue lines indicate that $C \neq C'$ is only based on Conjecture 5.14.*

Proof. The classes are separated from or subsets of each other as indicated in Table 5.1. The table cells contain either the name of a language that is in the class indicated by the row label but not in the class indicated by the column label or the symbol $\subseteq$ if no such language exists. A ? denotes the cases where the inclusion is based on Conjecture 5.14 and we thus do not know a separating language.

The entries with an orange background are proven explicitly in Lemmas 5.5 – 5.13. The entries with a yellow background follow from these with Corollary 3.7. The entry with blue background follows directly from Conjecture 5.14. The remaining inclusion results follow immediately from the definitions and the remaining seperability results follow from the proven results and inclusions. □

5.2. Relation to Other Classes

Having (mostly) settled the internal hierarchy of our language classes we now turn our attention to the relation to other classes, specifically the classes from the Chomsky hierarchy and the classes with restricted rewriting presented in Section 1.4.

We start by showing that $L_{ww} = \{ww \mid w \in \Sigma^*\}$ is in most of the classes from Section 1.4, but (at least for $|\Sigma| \geq 4$) not in [F][L][E][D]CFL. The latter follows from the fact that these languages do not satisfy the Interchange Lemma, as proven in [Yam08]. $L_{ww} \in$ CCFL is shown e.g. in [Kal10].

We now give a PRCG for L_{ww} for $\Sigma = \{a, b\}^*$, that can be straightforwardly extended to larger alphabets: $G_p = \{\{S, L, L_a, L_b, R, R'\},\ \{a, b\},\ R_p,\ S\}$, where

$$\begin{aligned} R_p = \{&(S \to LR;\ \emptyset)\ ,\ (L \to aL_a;\ \{R\}),\ (R \to aR';\ \{L_a\}),\ (L_a \to L;\ \{R'\}),\ (L \to \epsilon;\ \emptyset),\\ &(R' \to R;\ \{L\}),\ (L \to bL_b;\ \{R\}),\ (R \to bR';\ \{L_b\}),\ (L_b \to L;\ \{R'\}),\ (R \to \epsilon;\ \emptyset)\}. \end{aligned}$$

The conditions on the rules guarantee that we always replace the nonterminal symbols in the order L, R, L_x, R', and during each such cycle the same symbol is added to the left and right part.

Making use of the fact that at each point of the derivation (except the very beginning and end) there is exactly one of the symbols L, L_a, L_b and one of the symbols R, R', we can transform G_p into an equivalent FRCG $G_f = \{\{S, L, L_a, L_b, R, R', E\},\ \{a, b\},\ R_f,\ S\}$, where

$$\begin{aligned} R_f = \{&(S \to LR;\ \emptyset)\ ,\ (L \to aL_a;\ \{R', E\}),\ (R \to aR';\ \{L, L_b, E\}),\ (L_a \to L;\ \{R\}),\\ &(R' \to R;\ \{L_a, L_b\}),\ (L \to bL_b;\ \{R', E\}),\ (R \to bR';\ \{L, L_b, E\}),\ (L_b \to L;\ \{R\}),\\ &(L \to E;\ \emptyset)\ ,\ (R \to E;\ \emptyset)\ ,\ (E \to \epsilon;\ \{L, L_a, L_b, R, R'\})\}. \end{aligned}$$

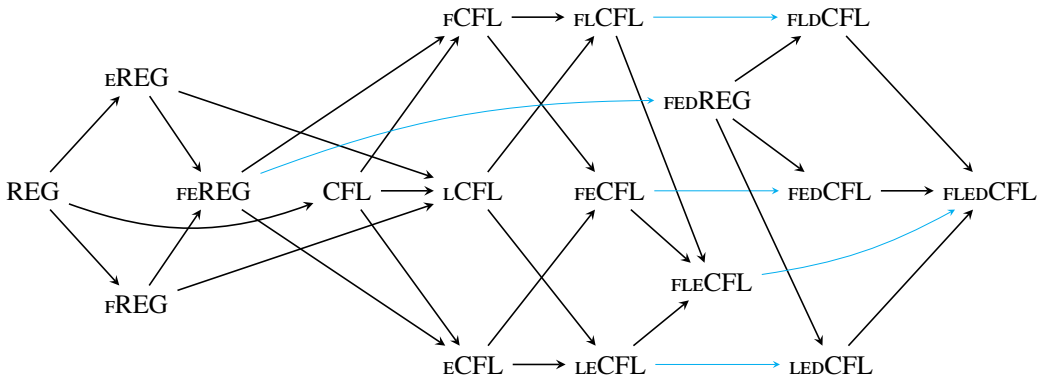


Figure 5.2.: Inclusion hierarchy of the languages considered in this thesis. An arrow (or path) from class C to class C' means $C \subsetneq C'$. Thinner blue lines indicate that $C \neq C'$ is only based on Conjecture 5.14. If there is no path, neither class is a subset of the other.

	FLEDCFL	FLDCFL	FEDCFL	LEDCFL	FLECFL	FLCFL	FECFL	LECFL	FCFL	LCFL	ECFL	CFL	FEDREG	FEREG	FREG	EREG	REG
FLEDCFL	$\subseteq$	L_4^R	L_3	L_4	?	L_4^R	L_3	L_4	L_3	L_4	L_3	L_3	L_D	L_D	L_D	L_D	L_D
FLDCFL	$\subseteq$	$\subseteq$	L_3	L_4	?	?	L_3	L_4	L_3	L_4	L_3	L_3	L_D	L_D	L_D	L_D	L_D
FEDCFL	$\subseteq$	L_4^R	$\subseteq$	L_4	?	L_4^R	?	L_4	L_4^R	L_4	L_4	L_4	L_D	L_D	L_D	L_D	L_D
LEDCFL	$\subseteq$	L_4^R	L_3	$\subseteq$	?	L_4^R	L_3	?	L_3	L_4^R	L_3	L_3	L_D	L_D	L_D	L_D	L_D
FLECFL	$\subseteq$	L_4^R	L_3	L_4	$\subseteq$	L_4^R	L_3	L_4	L_3	L_4	L_3	L_3	L_D	L_D	L_D	L_D	L_D
FLCFL	$\subseteq$	$\subseteq$	L_3	L_4	$\subseteq$	$\subseteq$	L_3	L_4	L_3	L_4	L_3	L_3	L_D	L_D	L_D	L_D	L_D
FECFL	$\subseteq$	L_4^R	$\subseteq$	L_4	$\subseteq$	L_4^R	$\subseteq$	L_4	L_4^R	L_4	L_4	L_3	L_D	L_D	L_D	L_D	L_D
LECFL	$\subseteq$	L_4^R	L_3	$\subseteq$	$\subseteq$	L_4^R	L_3	$\subseteq$	L_3	L_4	L_3	L_3	L_D	L_D	L_D	L_D	L_D
FCFL	$\subseteq$	$\subseteq$	$\subseteq$	L_4	$\subseteq$	$\subseteq$	$\subseteq$	L_4	$\subseteq$	L_4	L_4	L_4	L_D	L_D	L_D	L_D	L_D
LCFL	$\subseteq$	$\subseteq$	L_3	$\subseteq$	$\subseteq$	$\subseteq$	L_3	$\subseteq$	L_3	$\subseteq$	L_3	L_3	L_D	L_D	L_D	L_D	L_D
ECFL	$\subseteq$	L_4^R	$\subseteq$	$\subseteq$	$\subseteq$	L_4^R	$\subseteq$	$\subseteq$	L_4^R	L_4^R	$\subseteq$	L_4^R	L_D	L_D	L_D	L_D	L_D
CFL	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	L_D	L_D	L_D	L_D	L_D
FEDREG	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	?	?	?	?	?	L_2	?	L_5	$\subseteq$	?	L_5	L_5^R	L_5
FEREG	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	L_2	$\subseteq$	L_5	$\subseteq$	$\subseteq$	L_5	L_5^R	L_5
FREG	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	L_5^R	$\subseteq$	$\subseteq$	$\subseteq$	L_5^R	L_5^R
EREG	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	L_5	$\subseteq$	$\subseteq$	L_5	$\subseteq$	L_5
REG	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$

Table 5.1.: Separating languages. Each entry denotes that the respective language is in the class indicated by the row label but not in the class indicated by the column label. $\subseteq$ indicates that no such language exists, ? that we only conjecture the existence of such a language.

By the hierarchy depicted in Figure 1.1, these results already cover all of the classes except those generated by valence grammars and equivalent formalisms. For these, we have to use a different language to show that they are not contained in FLEDCFL.

Consider the language $\{w \in \{a_1b_1, \ldots, a_6b_6\}^* \mid |w|_{a_i} = \frac{1}{12}|w|, 1 \leq i \leq 6,\ |w| \in \{12\}\{2\}^{\mathbb{N}}\}$. It is generated by the $\mathbb{Z}^4$VG $G = (\{S\},\ \{a, b, c, d, e, f\},\ R,\ S)$, where

$$R = \{(S \to a_1b_1Sa_2b_2S;\ (1,1,1,1)),\ (S \to a_3b_3S;\ (-1,0,0,0)),\ (S \to a_4b_4S;\ (0,-1,0,0)),\\ (S \to a_2b_2Sa_1b_1S;\ (1,1,1,1)),\ (S \to a_5b_5S;\ (0,0,-1,0)),\ (S \to a_6b_6S;\ (0,0,0,-1)),\\ (S \to \epsilon;\ (0,0,0,0))\}.$$

However, in the proof of Lemma 7.7 we show that this language does not satisfy the Interchange Lemma and thus is no FLEDCFL.

It is reasonable to assume that the corresponding languages over smaller alphabets are examples for languages in $\mathbb{Z}^i$VL $\setminus$ FLEDCFL, $i < 4$. However, since these languages satisfy the Interchange Lemma, a formal proof of this fact is difficult.

Looking at the inverse direction, we easily note that the complexity of the languages we get depends on the complexity of the plsets we use.

If we allow arbitrary plsets, we can even generate languages that are not recursively enumerable, e.g. a^M for a non-RE set M. Obviously, this is of no practical use.

If we restrict ourselves to recursively enumerable (resp. decidable) plsets, we immediately get recursively enumerable (decidable) languages since now we can simulate e.g. one of the parsers from Chapter 4 on a Turing machine.

If we further restrict to plsets decidable in linear space with respect to the values being decided, we can decide membership using an LBA and thus get only context-sensitive languages.

A restriction of plsets that might be interesting in practice is to require that the parsers from Chapter 4 take the same asymptotic runtime as for traditional CFG. Note that this holds for all the grammars given in this thesis. With this restriction, we get a hierarchy of language classes that is embedded between the context-free and context-sensitive languages, in which each class is efficiently parsable and contains non-semilinear languages. A combination that is rather uncommon, e.g. none of the classes described in Section 1.4 has all of these properties.

We get another interesting result by restricting the plsets to semilinear sets. In this case, FLECFG can only generate context-free languages and FEREG = REG. This does not hold for FLEDCFG resp. FEDREG as can be seen from the FLDRLING G_1 in Example 2.5 that generates the non context-free language $a^n b^n c^n$ using only semilinear plsets.

Lemma 5.16. *If $G \in$ FLECFG and each plset in G is semilinear, then $L(G) \in$ CFL. If furthermore, G is a right linear or left linear grammar, $L(G) \in$ REG.*

Proof. Since $G \in$ FLECFG, each plset in G is of the form $M_f \times M_l \times M_e$ with semilinear sets M_f, M_l, and M_e. We will remove each of the components separately, starting with M_f.

By Theorem 3.16 there is an FLEPDA $A = (Q_A, \Sigma, \Gamma, \delta_A, q_{0,A}, g_0, F_A)$ with $L(A) = L(G)$ that uses the same plsets as G. Let $M = M_f \times M_l \times M_e$ be one of these plsets.

Since M_f is semilinear, $\Sigma^{M_f} \in$ REG and thus accepted by a DFA $B = (Q_B, \Sigma, \delta_B, q_{0,B}, F_B)$ without ϵ-transitions.

Now, the idea is to run B in parallel to A and whenever A would check if the current f-distance is in M_f we instead check if B is in an accepting state. Formally, let

$$C = (Q_A \times Q_B,\ \Sigma,\ \Gamma,\ \delta_C,\ (q_{0,A}, q_{0,B}),\ g_0,\ F_A \times Q_B),$$

with

$$\delta_C((q_A, q_B), a, g) = \Big\{((q'_A, q'_B), \gamma, M) \;\Big|\; (a \in \Sigma \wedge q'_B \in \delta_B(q_B, a)) \vee (a = \epsilon \wedge q'_B = q_B),\\ \Big((q'_A, \gamma, M) \in \delta_A(q_A, a, g) \;\wedge\; M \neq M_f \times M_l \times M_e)\\ \vee\; ((q'_A, \gamma, M_f \times M_l \times M_e) \in \delta_A(q_A, a, g)\\ \wedge\; M = \mathbb{N} \times M_l \times M_e \;\wedge\; q_B \in F_B)\Big)\Big\}.$$

Repeating this construction for all plsets, we get an lePDA that accepts $L(G)$.

By reversing the language, applying the above construction, and reversing again, we can further reduce the automaton to an lPDA.

Replacing the flePDA by an feNFA in the above construction already completes the proof for right/left linear grammars.

In order to eliminate length restrictions, we switch over to grammars. Let $G = (N, \Sigma, R, S)$ an lCFG in CNF where each plset is of the form $\mathbb{N} \times M_l \times \mathbb{N}$ with semilinear M_l. There are global constants n_0 and k so that for each of the M_l and all $n \geq n_0$: $n \in M_l \leftrightarrows n + k \in M_l$. ($k$ can be set to any common multiple of the periods of the linear sets, the plsets are comprised of; n_0 can be set to any value larger than all the base values of these sets.) W.l.o.g. we assume $n_0 \equiv 0 \mod k$ and $n_0 > 1$. (We can simply increase n_0 to the next such value.)

We use this fact to translate the lCFG into an equivalent CFG as follows: For $0 \leq i < n_0$ and $A \in N$, we introduce a symbol A_i with the intention that this symbol can be derived exactly to those (sub)words that can be derived from A and that have length i. Additionally, for $A \in N$ and $0 \leq j < k$ we introduce A'_i that will be derivable to words that have length $l \geq n_0$, $l \equiv i \mod k$.

In order to guarantee that each symbol derives exactly the desired words, we add the rule $(A_{i_1} \to B_{i_2} C_{i_3})$ only if $(A \to BC;\ \mathbb{N} \times M_l \times \mathbb{N})$ is a rule in the original grammar, $i_1 = i_2 + i_3$, and $i_1 \in M_l$. For symbols A'_{i_1} we analogously require $i_1 \equiv i_2 + i_3 \mod k$, and $n_0 + i_1 \in M_l$. We add rules for all combinations of B_{i_2}, B'_{i_2} and C_{i_3}, C'_{i_3}. Rules $(A_i \to a)$ are only needed for $i = 1$.

Finally, we add rules $(S \to S_i)$ resp. $(S \to S'_i)$ to get a single start symbol.

Formally, let $G' = (N', \Sigma, R', S)$, where

$$\begin{aligned}
N' &= \{S\} \cup \{A_i \;|\; A \in N, 0 \leq i < n_0\} \cup \{A'_i \;|\; A \in N, 0 \leq i < k\},\\
R' &= \{(S \to S_i) \;|\; 0 \leq i < n_0\} \cup \{(S \to S'_i) \;|\; 0 \leq i < k\}\\
&\quad \cup \{(A_{i_1} \to B_{i_2} C_{i_3}) \;|\; 0 \leq i_1 = i_2 + i_3 < n_0,\ (A \to BC;\ \mathbb{N} \times M_l \times \mathbb{N}) \in R,\ i_1 \in M_l\}\\
&\quad \cup \{(A'_{i_1} \to B'_{i_2} C'_{i_3}) \;|\; 0 \leq i_1, i_2, i_3 < k,\ (A \to BC;\ \mathbb{N} \times M_l \times \mathbb{N}) \in R,\\
&\qquad\qquad i_1 \equiv i_2 + i_3 \mod k,\ n_0 + i_1 \in M_l\}\\
&\quad \cup \{(A'_{i_1} \to B'_{i_2} C_{i_3}) \;|\; 0 \leq i_1, i_2 < k,\ 0 \leq i_3 < n_0,\ (A \to BC;\ \mathbb{N} \times M_l \times \mathbb{N}) \in R,\\
&\qquad\qquad i_1 \equiv i_2 + i_3 \mod k,\ n_0 + i_1 \in M_l\}\\
&\quad \cup \{(A'_{i_1} \to B_{i_2} C'_{i_3}) \;|\; 0 \leq i_1, i_3 < k,\ 0 \leq i_2 < n_0,\ (A \to BC;\ \mathbb{N} \times M_l \times \mathbb{N}) \in R,\\
&\qquad\qquad i_1 \equiv i_2 + i_3 \mod k,\ n_0 + i_1 \in M_l\}\\
&\quad \cup \{(A'_{i_1} \to B_{i_2} C_{i_3}) \;|\; 0 \leq i_1 < k,\ 0 \leq i_2, i_3 < n_0,\ (A \to BC;\ \mathbb{N} \times M_l \times \mathbb{N}) \in R,\\
&\qquad\qquad i_1 \equiv i_2 + i_3 \mod k,\ n_0 + i_1 \in M_l\}\\
&\quad \cup \{(A_1 \to a) \;|\; (A \to a;\ \mathbb{N} \times M_l \times \mathbb{N}) \in R,\ 1 \in M_l\}.
\end{aligned}$$

Obviously, $G' \in$ CFG. $L(G') = L(G)$ can be shown by a straightforward structural induction on the valid partial parse trees. □

As a final result on restricted plsets we note that if all plsets on rules replacing the start symbol are finite, we can only generate finite languages. This can easily be seen by noting that we can only generate words with a length that appears in one of those plsets. Thus, we are restricted to words with lengths from a finite set.

However, even if such FLEDCFG do not extend the generative power of CFG, they can be useful to get more concise descriptions of languages as can be seen in the following theorem. We will also make use of this feature for the application in Chapter 8.

Theorem 5.17. *For each $k \in \mathbb{N}$ there is a context-free (resp. regular) language L, that can be generated by an* FLE[D]CFG *(*FLE[D]RLING*) with m productions and finite plsets but cannot be generated by any* CFG *(*RLING*) with fewer than $k \cdot m$ productions.*

Proof. In [BMCW81] Bucher et. al. show that for each $n \in \mathbb{N}$ the language

$$U_n = \{\, a^k b^k c a^l b^l d a^m b^m \;\big|\; 0 \leq 2(k+l+m) \leq n-2 \}$$

needs $\Theta(n^2)$ context-free productions and the language

$$R_n = \{a^i b^j \;\big|\; 0 \leq i + j \leq n\}$$

needs $\Theta(n)$ regular productions.

U_n can however be generated by $G = (\{a, b, c, d, S, A\},\ \{a, b, c, d\},\ R,\ S)$ with

$$R = \{\, (S \to AcAdA;\ \{0, \ldots, n\}^3),\ (A \to aAb;\ \{0, \ldots, n\}^3),\ (A \to \epsilon;\ \{0, \ldots, n\}^3)\},$$

which has only 3 productions. R_n can be generated by $G = (\{a, b, S, B\},\ \{a, b\},\ R,\ S)$ with

$$R = \{(S \to aS;\ \{0, \ldots, n\}^3),\ (S \to B;\ \{0, \ldots, n\}^3),\ (B \to bB;\ \{0, \ldots, n\}^3),\ (B \to \epsilon;\ \{0, \ldots, n\}^3)\},$$

which has only 5 productions. □

6. Decidability Properties

We have already established in Chapter 4 that the membership problem is decidable for FLEDCFG, as long as all the plsets are decidable. For completeness let us repeat that result.

Theorem 6.1. *The membership problem, i.e. the question $w \in L(G)$, is decidable for any* FLEDCFG *with decidable plsets and any word w.* □

In order to show that many other interesting properties are undecidable for [F][L][E][D]CFG, we will show that they would solve the Post Correspondence Problem. Thus, let us repeat the definition of this problem.

Definition 6.2. *Let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ be two lists of nonempty words over a common alphabet Σ. The* Post Correspondence Problem (PCP) *is to determine, whether there exist $i_1, \ldots, i_k$ with $1 \leq i_j \leq n,\ 1 \leq j \leq k$, such that*

$$x_{i_1} \cdots x_{i_k} = y_{i_1} \cdots y_{i_k}.$$

Theorem 6.3 ([Pos46])**.** *For $|\Sigma| \geq 2$ the PCP is not uniformly decidable.*

We will not use Theorem 6.3 directly. Instead we reduce the PCP to the problem of deciding the emptiness of the intersection of two sets of natural numbers:

Lemma 6.4. *The property $M_1 \cap M_2 = \emptyset$ is not uniformly decidable for sets $M_1, M_2 \subseteq \mathbb{N}$, even if membership is decidable in logarithmic time for M_1 and M_2.*

Proof. Let $\Sigma = \{0, 1\}$ and $x, y \in (\Sigma^*)^n$ be an instance of the PCP.

For each sequence $\sigma = (i_1, \ldots, i_k) \in \{1, \ldots, n\}^k$, let

$$X_\sigma = x_{i_1} \cdots x_{i_k},\ Y_\sigma = y_{i_1} \cdots y_{i_k},\ \text{and } Z_\sigma = bin(i_1) \cdots bin(i_k),$$

where $bin(i)$ is the binary representation of i with $\lceil \log_2(n) \rceil$ digits (potentially including leading zeroes).

Now, define

$$M_1 = \{m \in \mathbb{N} \mid \exists \sigma\colon\ 2X_\sigma 2Z_\sigma 2 \text{ is ternary representation of } m\}$$

and

$$M_2 = \{m \in \mathbb{N} \mid \exists \sigma\colon\ 2Y_\sigma 2Z_\sigma 2 \text{ is ternary representation of } m\}.$$

For a given number m it is straightforward to decide in time linear in the length of the ternary representation of m, i.e. logarithmic time in its value, if m is in M_1 resp. M_2.

Furthermore, the parts Z_σ establish a one-to-one correspondence between sequences and words in M_1 resp. M_2. Thus, a number m is in $M_1 \cap M_2$, if and only if it can be written as $2X_\sigma 2Z_\sigma 2$ and $2Y_\sigma 2Z_\sigma 2$ for the same sequence σ. This, in turn, is equivalent to σ being a solution to the PCP for x and y, completing the reduction. □

This result already suffices to prove the main theorem of this section.

Theorem 6.5. *The following properties are not uniformly decidable for languages represented by grammars G_1 and G_2 from any of the classes* [F][L][E][D]CFG, [F][L][E][D]RLING, [F][L][E][D]LLING *with higher generative power than* CFL *resp.* REG. *This holds even if we only allow plsets that are decidable in logarithmic time.*

1. *Emptyness, i.e.* $L(G_1) \stackrel{?}{=} \varnothing$,
2. *universality, i.e.* $L(G_1) \stackrel{?}{=} \Sigma^*$,
3. *regularity, i.e.* $L(G_1) \stackrel{?}{\in} \text{REG}$,
4. *context-freeness, i.e.* $L(G_1) \stackrel{?}{\in} \text{CFL}$,
5. *equality, i.e.* $L(G_1) \stackrel{?}{=} L(G_2)$,
6. *inclusion, i.e.* $L(G_1) \stackrel{?}{\subseteq} L(G_2)$ *and*
7. *disjointness, i.e.* $L(G_1) \cap L(G_2) \stackrel{?}{=} \varnothing$.

Proof. We show undecidability explicitly for LRLING. For LLLING, they follow by Lemma 2.12, for FRLING and ELLING, they follow using the constructions in Lemmas 3.5 and 3.6. All other classes contain at least one of these classes as a subclass.

1. For $M_1, M_2 \subseteq \mathbb{N}$, let $G_{M_1,M_2} = (\{S, S', A\},\ \{a\},\ R,\ S)$ with
$$R = \{(S \to S';\ \mathbb{N} \times M_1 \times \mathbb{N}),\ (S' \to A;\ \mathbb{N} \times M_2 \times \mathbb{N}),\ (A \to aA;\ \mathbb{N}^3),\ (A \to \epsilon;\ \mathbb{N}^3)\}.$$
Obviously, $L(G_{M_1,M_2}) = \varnothing \ \leftrightarrows\ M_1 \cap M_2 = \varnothing$. Thus by Lemma 6.4, the claim follows.

2. For $M_1, M_2 \subseteq \mathbb{N}$ let $\overline{G}_{M_1,M_2} = (\{S, A\},\ \{a\},\ R,\ S)$ with
$$R = \{(S \to A;\ \mathbb{N} \times \mathbb{N} \backslash M_1 \times \mathbb{N}),\ (S \to A;\ \mathbb{N} \times \mathbb{N} \backslash M_2 \times \mathbb{N}),\ (A \to aA;\ \mathbb{N}^3),\ (A \to \epsilon;\ \mathbb{N}^3)\}.$$
Here we have $L(\overline{G}_{M_1,M_2}) = \Sigma^* \ \leftrightarrows\ M_1 \cap M_2 = \varnothing$ proving the claim.

3. Once more, for $M_1, M_2 \subseteq \mathbb{N}$ we let $G'_{M_1,M_2} = (\{S, A, A', A''\},\ \{a\},\ R,\ S)$ with
$$R = \{(S \to A;\ \mathbb{N} \times \mathbb{N}^{\{2\}} \times \mathbb{N})\ ,\ (A \to aA;\ \mathbb{N}^3)\ \ ,\ (A \to A';\ \mathbb{N} \times M_1 \times \mathbb{N}),$$
$$(A' \to A'';\ \mathbb{N} \times M_2 \times \mathbb{N}),\ (A'' \to aA'';\ \mathbb{N}^3),\ (A'' \to \epsilon;\ \mathbb{N}^3)\}.$$
$L(G'_{M_1,M_2})$ contains exactly the words a^m, where m is a square at least as large as the smallest number in $M_1 \cap M_2$. If this set is nonempty, then it is not semilinear and thus, the language is not context-free or regular. Thus, $L(G'_{M_1,M_2}) \in \text{REG} \ \leftrightarrows\ M_1 \cap M_2 = \varnothing$.

4. Here, the previous construction applies as well.

5. For G_2 a grammar with no rules, this property is equal to the first one. Thus, it is also undecidable.

6. Since $L(G_1) = L(G_2) \ \leftrightarrows\ L(G_1) \subseteq L(G_2) \wedge L(G_2) \subseteq L(G_1)$, undecidability follows from the previous property.

7. For G_2 a grammar that generates Σ^*, $L(G_1) \cap L(G_2) = \varnothing \ \leftrightarrows\ L(G_1) = \varnothing$. Since the latter is undecidable, the former is also undecidable.

□

The final three reductions use well-known ideas (cp. [HU79, Theorem 8.12]).

7. Closure Properties

We conclude the survey of our language classes by looking at their closure properties.

We start with some results for which the proofs from the traditional classes carry over. Specifically, the first two constructions are based on Corollary 1 from [Har78, Chapter 3.4].

Lemma 7.1. *The classes* [F][L][E][D]CFL *and* [F][E][D]REG *are closed under union.*

Proof. For $G_1 = (N_1, \Sigma, R_1, S_1)$ and $G_2 = (N_2, \Sigma, R_2, S_2)$ two [F][L][E][D]CFG, define

$$G = (N_1 \sqcup N_2 \sqcup \{S\},\ \Sigma,\ R_1 \cup R_2 \cup \{(S \to S_1;\ \mathbb{N}^3), (S \to S_2;\ \mathbb{N}^3)\},\ S).$$

It is easily verified that G is in the same class as G_1 and G_2 (and right resp. left linear, if both G_1 and G_2 are) and generates $L(G_1) \cup L(G_2)$. □

Lemma 7.2. *The classes* [L]CFL *are closed under concatenation and Kleene Star.*

Proof. For $G_1 = (N_1, \Sigma, R_1, S_1)$ and $G_2 = (N_2, \Sigma, R_2, S_2)$ two [L]CFG, define

$$G = (N_1 \sqcup N_2 \sqcup \{S\},\ \Sigma,\ R_1 \cup R_2 \cup \{(S \to S_1 S_2;\ \mathbb{N}^3)\},\ S).$$

It is easily verified that G is an [L]CFG and $L(G) = L(G_1) \cdot L(G_2)$.

For $G = (N, \Sigma, R, S) \in$ [L]CFG,

$$G' = (N \sqcup \{S'\},\ \Sigma,\ R \cup \{(S' \to S'S;\ \mathbb{N}^3), (S' \to \epsilon;\ \mathbb{N}^3)\},\ S')$$

is a [L]CFG generating $L(G)^*$ □

The following contruction can be found e.g. in [Neb12, page 51].

Lemma 7.3. *The classes* [F][E][D]REG *are closed under complement and intersection.*

Proof. By replacing F with $Q \setminus F$ in a complete [F][E][D]DFA without ϵ-transitions, we get an automaton of the same type that accepts the complement of the original language.

Since $A \cap B = \overline{\overline{A} \cup \overline{B}}$, this together with Lemma 7.1 implies that the regular-based classes are also closed under intersection. □

Lemma 7.4. *The classes* [F][L][E][D]CFL *and* [F][E][D]REG *are closed under intersection with a regular set.*

Proof. For the regular-based classes, this is already implied by Lemma 7.3.

For the context-free-based classes, let $A = (Q_A, \Sigma, \Gamma, \delta_A, q_{0,A}, g_0, F_A)$ be an [F][L][E][D]PDA and $B = (Q_B, \Sigma, \delta_B, q_{0,B}, F_B)$ be an DFA without ϵ-transitions.

Then we can construct an [F][L][E][D]PDA C with $T(C) = T(A) \cap T(B)$ by the well-known product construction ([Har78, Theorem 6.4.1]):

$C = (Q_A \times Q_B,\ \Sigma,\ \Gamma,\ \delta_C,\ (q_{0,A}, q_{0,B}),\ g_0,\ F_A \times F_B)$, with

$$((q'_A, q'_B), \gamma, M) \in \delta_C((q_A, q_B), a, g) \rightsquigarrow (q'_A, \gamma, M) \in \delta_A(q_A, a, g) \wedge q'_B \in \delta_B(q_B, a) \text{ and}$$
$$((q'_A, q'_B), \gamma, M) \in \delta_C((q_A, q_B), \epsilon, g) \rightsquigarrow (q'_A, \gamma, M) \in \delta_A(q_A, \epsilon, g) \wedge q'_B = q_B.$$

It is straightforward to verify that C has the desired properties. □

Lemma 7.5. *The classes* [F][L][E][D]CFL *are not closed under intersection or complement.*

Proof. Let $L_1 = \{vcv^Rcw \mid v, w \in \{a, b\}^*\}$ and $L_2 = \{wcv^Rcv \mid v, w \in \{a, b\}^*\}$. Obviously $L_1, L_2 \in$ CFL. We show that $L = L_1 \cap L_2 = \{vcv^Rcv \mid v \in \{a, b\}^*\}$ does not satisfy the Interchange Lemma and thus none of the context-free-based classes are closed under intersection.

For $n = 3n' + 2$, $n' \in \mathbb{N}$, let $Q_n = L \cap \Sigma^n$ and $m = 2n' + 2$. For each $z_i = v_icv_i^Rcv_i \in Q_n$, the Interchange Lemma implies a decomposition $z_i = w_ix_iy_i$, where $\frac{m}{2} < |x_i| \leq m$. By our choice of m, this implies $|w_i| + |y_i| \geq n' = |v_i|$. Since w_i is a prefix of v_i, and y_i is a suffix of v_i, these parts are already sufficient to uniquely determine v_i and thus z_i. Thus $w_ix_jy_i \in L$ if and only if $x_j = x_i$. In order to satisfy the Interchange Lemma, we would thus have to find a subset of Q_n of size

$$k \geq \frac{|Q_n|}{c_Ln^2},$$

such that the part x_i and its position is identical for all words in the subset. We will now show that there is no such set.

If we choose $|x_i| = \lfloor \frac{m}{2} \rfloor + 1 = n' + 2$ and position x_i such that one of the letters c is exactly in the middle of x_i, x_i will determine slightly more than half of the letters of v_i. For larger x_i or other positions within the word, this fraction will obviously be larger.

Thus for each possible value of x_i, there are less than $2^{n'/2}$ words z_i in Q_n that share this value and position of x_i. Since $|Q_n| = 2^{n'}$, the Interchange Lemma would require that

$$2^{\frac{n'}{2}} \geq \frac{2^{n'}}{c_L \cdot (3n')^2},$$

which is equivalent to

$$1 \geq \frac{2^{\frac{n'}{2}}}{9c_L \cdot n'^2}.$$

In the latter equation the right-hand side tends to ∞ as n' gets large, proving our claim.

Since the classes are closed under union, but not under intersection and $A \cap B = \overline{\overline{A} \cup \overline{B}}$, they can not be closed under complement. □

Lemma 7.6. *The classes* [F][L][E][D]CFL *and* [F][E][D]REG *are closed under length-preserving homomorphism.*

Proof. By applying the homomorphism h to the rules of a grammar for L, we get a grammar for $h(L)$. Since the homomorphism is length-preserving, this does not interfere with the restrictions. □

Obviously, this simple construction (that is also based on [Har78, Chapter 3.4]) no longer works for homomorphisms that do not preserve length. It can be saved for homomorphisms where all symbols are replaced by a string of the same length c, if we multiply all positions and lengths in the plsets by c.

For arbitrary homomorphisms however, the closure property no longer holds, as we will now show.

Lemma 7.7. *The classes* [F][L][E][D]CFL *and* [F][E][D]REG*, with the exception of* CFL *and* REG*, are not closed under ϵ-free homomorphism, arbitrary homomorphism or inverse homomorphism.*

Proof. The straightforward way to prove this lemma would be to give a language L that is in the classes and a homomorphism h, so that $h(L)$ resp. $h^{-1}(L)$ is not in the classes. Unfortunately, we have not been able to find a pair (L, h) for which we could prove these properties.

Instead, we use a multi-step construction. Starting with the regular language $L_0 = \{a_1, \ldots, a_6\}^*$, we will construct in 6 steps the language

$$L_6 = \{w \in \{a_1b_1, \ldots, a_6b_6\}^* \;\middle|\; |w|_{a_i} = \frac{1}{12}|w|, 1 \leq i \leq 6,\ |w| \in \{12\} \cdot \{2\}^{\mathbb{N}}\}.$$

In step i, $1 \leq i \leq 6$, we define $L_i = h_i(L_{i-1} \cap \Sigma^{M_i})$, where h_i replaces a_i by a_ib_i, leaving all other characters unaffected, $\Sigma = \{a_i, b_i \mid 1 \leq i \leq 6\}$, and $M_i = \{1 + \frac{i-1}{6}\} \cdot \{6\} \cdot \{2\}^{\mathbb{N}}$. Intersecting with Σ^{M_1} ensures that

- we keep words that can contain an equal number of all the characters, i.e. that have a length that is a multiple of 6, and
- the lengths of the words differ by a factor of at least 2. This way, the resulting length after applying each homomorphism uniquely determines the original length and the number of characters that were added by the homomorphism.[1]

The intersections with Σ^{M_i}, $i > 1$, then eliminate all words for which the previous homomorphism did not replace exactly $\frac{1}{6}$ of the characters of the original word, i.e. the number of characters a_{i-1} was not as required.[2]

With this construction, showing $L_6 \notin$ FLEDCFL gives the desired result, since all the classes included in the lemma are closed under intersection with Σ^M for arbitrary $M \subseteq \mathbb{N}$:

- If $L \in$ [F]L[E][D]CFL (resp. [F]L[E][D]RLINL or [F]L[E][D]LLINL)[3], we may add a new start symbol S' and the rule $(S' \to S;\ \mathbb{N} \times M \times \mathbb{N})$ to a grammar that generates L.
- If $L \in$ [F][L]E[D]CFL, we may add a new start symbol S', another new nonterminal E, and the rules $(S' \to ES;\ \mathbb{N}^3)$ and $(E \to \epsilon;\ \mathbb{N}^2 \times M)$ to a grammar that generates L.
- Finally, if $L \in$ F[L][E][D]CFL we may add a new start symbol S', another new nonterminal E, and the rules $(S' \to SE;\ \mathbb{N}^3)$ and $(E \to \epsilon;\ M \times \mathbb{N}^2)$ to a grammar that generates L.

In order to show that L_6 does not satisfy the Interchange Lemma, let $Q_n = L_6 \cap \Sigma^n$ for each $n = 12n'$, $n' \in \{2\}^{\mathbb{N}}$, and let $m = \frac{1}{3}n$. By the considerations above, the words in Q_n make up all permutations of the multiset consisting of n' copies of each pair $a_k b_k$. Using elementary combinatorics and Stirling's approximation, we find

$$|Q_n| = \binom{6n'}{n', n', n', n', n', n'} \sim \frac{\sqrt{6}}{\sqrt{\pi n'}^5} 6^{6n'}.$$

In order to show that L_6 does not satisfy the conditions of the Interchange Lemma, we need to get a bound on $|Q_n^{(i)}|$ for each word $z_i \in Q_n$, where

$$Q_n^{(i)} = \{z_j \in Q_n \mid |w_j| = |w_i|,\ |y_j| = |y_i|,\ w_i x_j y_i \in L_6\},$$

and $z_i = w_i x_i y_i$ is the partitioning according to the Interchange Lemma.

Since the number of occurences of each of the strings $a_k b_k$, $1 \le k \le 6$, is the same for each word in L_6, $z_j \in Q_n^{(i)}$ if and only if $|x_j|_{a_i} = |x_i|_{a_i}$ and the positions of x_i and x_j are the same.[4] Now, if x_i consists of n_k instances of $a_k b_k$, $1 \le k \le 6$, we get

$$|Q_n^{(i)}| = \binom{|x_i|/2}{n_1, n_2, n_3, n_4, n_5, n_6} \cdot \binom{6n' - (|x_i|/2)}{n' - n_1, n' - n_2, n' - n_3, n' - n_4, n' - n_5, n' - n_6},$$

where the first factor describes the number of permutations of the pairs in x_i and the second factor describes the number of permutations of the pairs in w_i and y_i. Within the bounds on $|x_i|$ allowed by our choice of m, the product of these terms is maximized for $|x_i| = 4n' = \frac{1}{3}n$ and $n_k = \frac{1}{3}n'$, $1 \le k \le 6$. Thus with Stirling's Approximation, the inequality from the Interchange Lemma becomes

$$\frac{3^6}{\sqrt{2}^{13}(\pi n)^5} 6^{6n} \ge \frac{\sqrt{6}}{c_{L_6}\sqrt{\pi n}^5 n^2} 6^{6n},$$

[1] We can not distinguish the case where all characters have been replaced by a pair of characters from that where no character has been replaced in a word twice as long. Since we want to eliminate the word in both cases, this is not a problem.

[2] We do not need to check the number of characters a_6 since it is already determined by the number of the other characters and the legnth of the original word.

[3] By Corollary 3.7, this covers all the regular-based classes included in the lemma.

[4] For simplicity we assume that $|w_i|$, $|x_i|$ and $|y_i|$ are even, so no pair $a_k b_k$ is split between them.

which is invalid for

$$n > \frac{3^{11} c_{L_6}^2}{2^{14}\pi^5}.$$

Thus, none of the classes is closed under ϵ-free homomorphism or arbitrary homomorphism. For the case of inverse homomorphism, note that $L_i = h_i'^{-1}(L_{i-1} \cap \Sigma^{M_i}) \cap L_i'$, where h_i' deletes b_i, leaving all other characters unaffected, and L_i' is the regular language $\{a_1 b_1, \ldots, a_i b_i, a_{i+1}, \ldots, a_6\}^*$. □

Another closure property that does not carry over to all classes is closure under concatenation. Intuitively, this fails because concatenating a language at one side causes loss of the "point of reference" for distance restrictions relative to that side. Thus, language features that rely on such checks can not be maintained after concatenation.

The same abstract argument also applies to languages that result from the Kleene star operation. Thus, it is not surprising that non-closure under this operation can be shown using only slight modifications of the proof for concatenation.

Lemma 7.8. *The classes* [F][L][E][D]CFL *and* [F][E][D]REG*, with the exception of* [L]CFL *and* REG *are not closed under concatenation or Kleene star.*

Proof. As in the previous proof, we have to resort to a multi-step construction in order to obtain a language that does not satisfy the Interchange Lemma. We will give the construction explicitly for F[E][D]REG and F[L][E][D]CFL. For [L]E[D]CFL and EREG, the result follows by reversing the construction.

We start with the regular language $L_0 = \{ba^*\}$ and in step i, $1 \leq i \leq 3$, define $L_i = L_0 \cdot f_i(L_{i-1})$, where f_i marks every character that appears at an f-distance that is a square, using a different marking for each i.

To see that the classes are closed under application of f_i, let $G \in$ F[L][D]RLING $\cup$ F[L][E][D]CFG. W.l.o.g. assume that the right-hand side of each rule in G contains either a single terminal symbol as the first character or no terminal symbols at all. (This holds e.g., if G is created by the construction in Theorem 3.5 or is in CNF.) We replace each rule $(A \to a\alpha;\ M)$, $a \in \Sigma$, $\alpha \in (N \cup \Sigma)^*$ in G, by two rules $(A \to a'\alpha;\ M \cap (\mathbb{N}^{\{2\}} \times \mathbb{N}^2))$ and $(A \to a\alpha;\ M \cap ((\mathbb{N} \setminus \mathbb{N}^{\{2\}}) \times \mathbb{N}^2))$, where a' is the marked version of a. This way we get a grammar G' in the same class as G with $L(G') = f_i(L(G))$.

We now show that the classes are not closed under concatenation by showing that L_3 does not satisfy the Interchange Lemma and thus can not be in [F][L][E][D]CFG.

Let $Q_n = \{ba^{n_1}ba^{n_2}ba^{n_3}ba^{n-n_1-n_2-n_3-4} \mid n_1, n_2, n_3 < \frac{1}{10}n\}$, the letters marked as implied by the construction of L_3.

Each subword of the final block of a's with a length of at least $2\sqrt{n}$ necessarily contains at least 2 instances of each type of marking. Since the distance between two neighbored squares uniquely determines the squares, such a subword is sufficient to reconstruct the positions of the b's and thus uniquely determines the complete word.

Now let $m = \frac{2}{3}n$, and consider $z_i = w_i x_i y_i$ and $z_j = w_j x_j y_j \in Q_n$ partitionings according to the Interchange Lemma, $|w_i| = |w_j|$, $|y_i| = |y_j|$. We show that $w_i x_j y_i \in L_3 \leftrightarrows i = j$:

By our choice of m, $|x_j| > \frac{1}{3}n$. Thus, x_j contains at least $\frac{1}{3}n - \frac{3}{10}n = \frac{1}{30}n$ characters from the final block of a's. Since also $|w_i| + |y_i| > \frac{1}{3}n$, at least one of these words will contain at least $\frac{1}{2} \cdot \frac{1}{30}n = \frac{1}{60}n$ characters from the final block of a's. Since for $n > 120^2$, $\frac{1}{60}n > 2\sqrt{n}$, x_j as well as the pair w_i, y_i uniquely determine the word they are from, implying $w_i x_j y_i \in L_3 \leftrightarrows i = j$.

Since $|Q_n| = \left(\frac{1}{10}n\right)^3$, this makes the inequality from the Interchange Lemma $1 \geq \frac{n^3}{1000 c_{L'} \cdot n^2} = \frac{n}{1000 c_{L'}}$, which is not satisfied for $n > 1000 c_L$.

To disprove closure under Kleene Star, we modify the construction as follows. Replace L_i by $L_i' = (f_i(L_i))^*$, $1 \leq i \leq 3$ and let

$$Q_n = \{bbba^{n_1}bbba^{n_2}bba^{n_3}ba^{n-n_1-n_2-n_3-9} \mid n_1, n_2, n_3 < \tfrac{1}{10}n\},$$

the letters marked as in $f_3(f_2(f_1(L))) \cdot f_3(f_2(f_1(L)) \cdot f_2(f_1(L) \cdot f_1(L))) \subset L_3'$. Then we still have $w_i x_j y_i \in L' \leftrightarrows i = j$ since the markings in the final block of a's are the same as above and thus combining parts from two words would still give an inconsistently marked block of a's.

	FLEDCFL	FLDCFL	FEDCFL	LEDCFL	FLECFL	FLCFL	FECFL	LECFL	FCFL	LCFL	ECFL	CFL	FEDREG	FEREG	FREG	EREG	REG
union	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
intersection	−	−	−	−	−	−	−	−	−	−	−	−	+	+	+	+	+
int. with regular set	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
complement	−	−	−	−	−	−	−	−	−	−	−	−	+	+	+	+	+
concatenation	−	−	−	−	−	−	−	−	−	+	−	+	−	−	−	−	+
Kleene star	−	−	−	−	−	−	−	−	−	+	−	+	−	−	−	−	+
homomorphism	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	+
ϵ-free hom.	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	+
length-preserving hom.	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
inverse hom.	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	+
reversal	+	−	+	−	+	−	+	−	−	+	−	+	+	+	−	−	+

Table 7.1.: Closure properties of the language classes examined in this thesis. A "+" indicates that the class is closed under the respective operation, a "–" indicates that it is not.

Thus, the above argument still applies. □

We close this chapter by combining all these results and Lemma 2.12 into a single table.

Theorem 7.9. *The closure properties listed in Table 7.1 hold.* □

8. Application: Prediction of RNA Secondary Structures

After looking at the properties of the language classes defined by position-and-length-dependent context-free grammars, we will now turn our attention to an application of length-dependent grammars in bioinformatics. As we will see, this application does not profit from ʟCFG because of their greater expressive power but because they allow for a more concise description of some context-free languages.

8.1. Problem Setting

Single-stranded RNA molecules consist of a sequence of nucleotides connected by phosphordiester bonds. Nucleotides only differ by the bases involved, them being adenine, cytosine, guanine and uracil. The sequence of bases is called the *primary structure* of the molecule and is typically denoted as a word over the alphabet $\{A, C, G, U\}$. Additionally, pairs of the bases can form hydrogen bonds[1], thus folding the molecule to a complex three-dimensional layout called the *tertiary structure*.

As determining the tertiary structure is computationally complex, it has proven convenient to first search for the *secondary structure*, for which only a subset of all hydrogen bonds is considered, so that the molecule can be modeled as a planar graph. Additionally, so-called pseudoknots are eliminated, that is, there is no subsequence "first base of (hydrogen) bond 1 ... first base of (hydrogen) bond 2 ... second base of bond 1 ... second base of bond 2" when traversing along the primary structure. See Figure 8.1 for an example of a secondary structure.

When abstracting from the primary structure, secondary structures are often denoted as words over the alphabet $\Sigma = \{(, |,)\}$, a corresponding pair of parentheses representing a pair of bases connected by a hydrogen bond, while a | stands for an unpaired base. For example, when starting transcription at the marked end, the structure from Figure 8.1 would be denoted by the word

(((((((||(((||||||||||||)))|(((((|||||||)))))|||||(((((|||||||)))))))))))||||.

The oldest and most commonly used method for computing the secondary structure is to determine the structure with minimum free energy. This was first done by Nussinov et al. who, based on the observation that a high number of paired bases corresponds to a low free energy, used dynamic programming to find the structure with the highest number of paired bases in cubic time ([NPGK78]).

While the energy models used today are much more sophisticated, taking into account, e.g. the types of bases involved in a pair or the types of bases located next to them, the dynamic programming scheme has remained the same (e.g. [ZS81]).

A different approach is based on probabilistic modeling. An (ambiguous) stochastic context-free grammar (SCFG) that generates the primary structures is chosen such that the parse trees for a given primary structure uniquely correspond to the possible secondary structures. The probabilities of this grammar are then trained either from molecules with known secondary structures or by an expectation maximization algorithm ([KH99]).

The probabilities on the parse trees as computed from the trained probabilities will model the probability distribution of the corresponding secondary structures, assuming the training data was representative and the grammar is actually capable of modeling this distribution. Thus, the most probable secondary structure (parse tree) is used as prediction ([KH99]).

[1] Typically adenine pairs with uracil and guanine pairs with cytosine. Other pairs are less stable and hence less common.

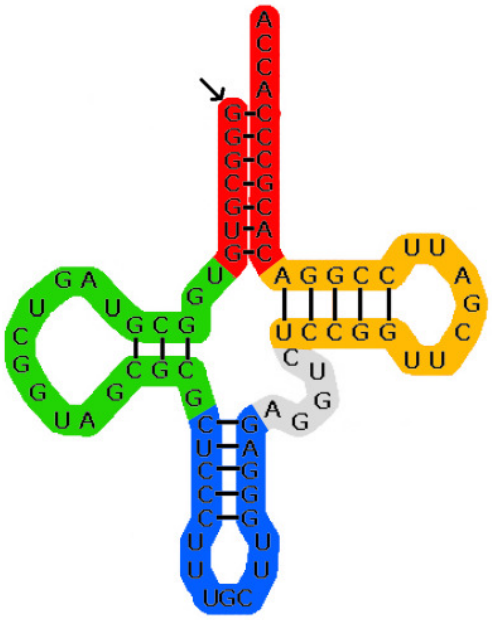


Figure 8.1.: Example of an RNA secondary structure. Letters represent bases, the colored band marks the phosphordiester bonds, short edges mark hydrogen bonds. The arrow marks the end at which we start reading the sequence. (The different colors only serve to identify the corresponding parts in the formal language representation.)

Many other approaches as well as extensions and modifications of the ones mentioned above have been suggested over the years. A recent overview can be found in [And10].

With the exception of approaches that simulate the actual physical folding process (e.g. [XBI05]), none of the established prediction algorithms take into account that the molecules grow sequentially and the folding takes place during their development. It has, however, been shown that this *co-transcriptional folding* has an effect on the resulting secondary structures (e.g. [BRK80, MM04]).

Since simulation algorithms have the downside of being computationally expensive, it is desirable to include the effects of co-transcriptional folding in the traditional algorithms. In the following, we will do so for the SCFG-approach by adding length-dependencies. Due to co-transcriptional folding, one would expect that the probability of two bases being paired depends on how far the bases are apart, and the probability of a part of the molecule forming a specific motif depends on how large that part of the molecule is. In the SCFG-model, the distance between two paired bases (resp. the size of a motif) is simply the size of the subword that results from the rule application introducing the base pair as first and last symbol (resp. starting to build the motif), assuming such a rule application exists. Thus, in order to model this variability of the probabilities, we will use length-dependent SCFG (ʟSCFG), which combine the ideas of SCFG and ʟCFG such that the probabilities are now assigned to pairs (rule, length of application).

8.2. Formal Definitions

We start by citing the definition of non-length-dependent SCFG.

Definition 8.1 ([Gre67])**.** *A* stochastic context-free grammar (SCFG) *is a 4-tuple* $G = (N, \Sigma, R, S)$*, where* N*,* Σ*, and* S *are defined as for* CFG*, and the rules in* R *are of the form* $(A \to \alpha;\ p)$*,* $p \in [0, 1]$*, so that for each*

$A \in N$ the sum of rule probabilities

$$\sum_{r \in R_A} p_r = 1.$$

Words are generated as for usual context-free grammars. The probability of a specific partial derivation $A \;=\!\![\Delta]\!\!\Rightarrow^* \alpha$ is defined by

$$P_G(\Delta) = \prod_{r \in \Delta} p_r.$$

Abstracting from the rules involved, we define the probability that A derives to α by

$$P_G(A \Rightarrow^* \alpha) = \sum_{A =[\Delta;L]\Rightarrow^* \alpha} P_G(\Delta).$$

Finally, for a word $w \in L(G)$, we define its probability by

$$P_G(w) = P_G(S \Rightarrow^* w).$$

G is called consistent *if and only if*

$$\sum_{w \in L(G)} P_G(w) = 1,$$

i.e. *if it defines a probability distribution on $L(G)$. Throughout this chapter, we will talk of probabilities on words and derivations even for (possibly) inconsistent grammars where there is no (guaranteed) probability distribution and the respective values thus would be more appropriately called weights.*

As usual, we will write P instead of P_G, if the grammar we are talking about is unambiguous from the context.

Now the straightforward way to make the rule probabilities depend on length is to make the rule probability p a function $p\colon \mathbb{N} \to [0, 1]$, the parameter being the length of the rule application in question, and change the constraint on the probabilities to

$$\forall A \in N,\ l \in \mathbb{N}\colon \sum_{r \in R_A} p_r(l) = 1.$$

We will in essence use this idea, but in order to ensure that our modified versions of the training algorithms provide consistent grammars – like the original versions do for SCFG – we need some technical additions.

First, we note that some combinations of rule and length can never occur in a derivation, e.g., a rule $(A \to \epsilon)$ with length 1. It is reasonable to set the probability of such combinations to 0, which is what the training algorithms will do. This requires us to relax the condition to

$$\forall A \in N,\ l \in \mathbb{N}\colon \sum_{r \in R_A} p_r(l) \in \{0, 1\},$$

in order to cover symbols for which no production can lead to a word of length l.

Further problems arise from the fact that, while for SCFG the probability in $(A \to \alpha;\ p)$ expresses the probability that a parse tree with root A starts with rule $(A \to \alpha)$, now $p(l)$ expresses the probability that a parse tree with root A starts with rule $(A \to \alpha)$, *given that* it generates a word of length l, but our model currently does not include the probabilities that a parse tree generates a word of a given length.

As an example consider the grammar characterized by the rules

$$r_1 = (S \to AS;\ p_1),\ r_2 = (A \to aA;\ p_2) \text{ and } r_3 = (A \to \epsilon;\ p_3).$$

Here, for each combination of nonterminal and length only one rule is applicable, leading to $p_i(l) = 1$ if r_i can lead to a word of length l, and $p_i(l) = 0$ if it cannot. Thus, for this grammar each parse tree will have "probability" 1 and each word will have a "probability" equal to its degree of ambiguity.

Checking the grammar more closely, we find two points where the length of the word generated from a nonterminal symbol is not determined when the symbol is introduced during a derivation: At the start of the

derivation the length of the generated word is not predetermined and when the rule $(S \to AS)$ is applied to generate a word of length l, there are $l + 1$ possible ways to distribute the length amongst the two symbols on the right-hand side.

Both cases can be countered by adding proper probability distributions to the model. However, while handling the start of derivations requires a single probability distribution P_ℓ on the natural numbers, the distribution of the length amongst the symbols on the right-hand side of a rule requires for each length l a set of probabilities (generally of size $O(l^{i-1})$ for a rule with i nonterminal symbols on the right-hand side). Obviously, storing these values with the grammar explicitly is not feasible in practice unless we only want to consider short words. Approximating them with easily describable functions seems possible, but would potentially be difficult to automate and would not be covered by the considerations in Section 8.3, regarding the consistency of probabilities gathered by expectation maximization. Thus, we decided to use uniform distributions here.

Combining these considerations, we define length-dependent stochastic context-free grammars:

Definition 8.2. *A* length-dependent stochastic context-free grammar (lSCFG) *is a 5-tuple* $G = (N, \Sigma, R, S, P_\ell)$*, where* N*,* Σ*, and* S *are defined as for* CFG*, the rules in* R *are of the form* $(A \to \alpha;\ p)$*,* $p\colon\ \mathbb{N} \to [0, 1]$ *a rule probability function, so that*

$$\forall A \in N,\ l \in \mathbb{N}\colon \quad \sum_{r \in R_A} p_r(l)\ \in \{0, 1\},$$

and $P_\ell\colon\ \mathbb{N} \to [0, 1]$ *a probability function on the lengths of words in* $L(G)$ *such that*

$$\sum_{l \in \mathbb{N}} P_\ell(l) = 1.$$

For $(A \to \alpha;\ p_r) \in R$ *and* $l \in \mathbb{N}$*, we define* $p_{\alpha,l} = \frac{1}{c_{\alpha,l}}$*, where* $c_{\alpha,l}$ *is the number of different assignments of lengths to the symbols of* α *that satisfy:*

- *Terminals are always assigned a length of* 1*.*
- *Any nonterminal symbol* B *is assigned a length* i *for which* $\exists w \in \Sigma^i\colon\ B \Rightarrow^*_{G'} w$*, where* G' *is the* CFG *that results from removing all probabilities from* G*.*
- *The assigned lengths add up to* l*.*

If $c_{\alpha,l} = 0$*, we define* $p_{\alpha,l} = 0$*.*

Words are generated as for usual context-free grammars. We define the probability of a specific partial derivation $A \Rightarrow\!\!\!\!\!\left[\begin{smallmatrix}\Delta=(A_1\to\alpha_1;\ p_1),\ldots,(A_k\to\alpha_k;\ p_k)\\ (f,l,r)\end{smallmatrix}\right]\!\!\!\!\Rightarrow \alpha$ *by*

$$P_G(\Delta, l) = \prod_{i=1}^{k} \left(p_i(l_i) \cdot p_{\alpha_i, l_i}\right),$$

where l_i *is the length of the* i*-th rule application in* Δ*, as defined in Definition 2.2. As in Definition 8.1, we define the probability that* A *derives to* α *by*

$$P_G(A \Rightarrow\!\!\!\![(f, l, r)]\!\!\!\!\Rightarrow^* \alpha) = \sum_{A \Rightarrow\!\!\!\![\Delta; L]\!\!\!\!\Rightarrow^* \alpha} P_G(\Delta, l).$$

Finally, for a word $w \in L(G)$*, we define its probability by*

$$P_G(w) = P_\ell(|w|) \cdot P_G(S \Rightarrow^* w).$$[2]

[2] Remember that we may omit the position/length-triple, if it is $(0, |w|, 0)$. Since the grammars in this chapter are not position-dependent, we get $\forall f, e \in \mathbb{N}\colon A \Rightarrow\!\!\!\![(f, |w|, e)]\!\!\!\!\Rightarrow^* w \ \leftrightarrows\ A \Rightarrow\!\!\!\![(0, |w|, 0)]\!\!\!\!\Rightarrow^* w$, allowing us to use the shorter notation extensively.

As before, G is called consistent *if and only if*

$$\sum_{w \in L(G)} P_G(w) = 1.$$

Also as before, we will drop the index $_G$ on the probabilities, if it is clear from the context.

This definition allows for the productions to be equipped with arbitrary probabilities as long as for any given pair of nonterminal and length, they represent a probability distribution or are all 0. In our experiments in Section 8.5, we will however confine ourselves with grouping the lengths together in finitely many intervals, a rule having the same probability for lengths that are in the same interval. This allows us to store the probabilities of each rule as a vector and retrieve them in the algorithms without further computation.

From Lemma 5.16, we can conclude that we can only model context-free languages this way. This is, however, not a problem since we know that all secondary structures without pseudoknots can be modeled by context-free grammars.

As we will see, however, in the following section, lSCFG with length intervals allow for a much more concise description of the probability distribution than an SCFG that captures the same amount of detail.

In the following sections, we will consider the two cases, arbitrary probabilities and finitely many intervals, mostly in parallel.

Note that even the simple class of structures containing at most a single pseudoknot of the form $({}^n[{}^m)^n]^m$ is not captured by fledCFG, as can be shown using the Interchange Lemma. Thus the restriction of expresiveness resulting from the use of intervals is not relevant in practice.

8.3. Estimating Rule Probabilities

When given a set of training data $\mathcal{T}$ of either full parse trees of the grammar or words from the language generated by it, we want to train the grammar according to the maximum-likelihood principle, i.e. choose rule probabilities such that

1. the grammar is consistent and
2. the likelihood $L(\mathcal{T}, P) = \prod_{t \in \mathcal{T}} P(t)$ is maximized among all sets of probabilities that satisfy 1.

For (non-length-dependent) SCFG, it is well-known how this can be achieved [Pre03, CG98]:

- If training from full parse trees, the relative frequencies with which the rules occur (among all rules with the same left-hand side) are a maximum-likelihood estimator.
- When given a training set of words from $L(G)$, the rule probabilities can be estimated using the inside-outside algorithm. This algorithm roughly works as follows:

  Given an initial estimate of the rule probabilities, it uses a modified version of a parsing algorithm to determine the total probability of all full parse trees for w that derive $w_{i,j}$ in a subtree starting with r, for each word w in the training set, each subword $w_{i,j}$ of w, and each rule $r \in R$. Summing up these probabilities for all subwords, it gets expected numbers of rule occurrences from which expected relative frequencies can be computed as in the case of given parse trees. By using these expected relative frequencies as a new estimate for the rule probabilities and iterating the procedure, we are guaranteed to converge to a set of rule probabilities that gives a (local) maximum likelihood.

Both algorithms can easily be converted for lSCFG: Instead of determining the (actual resp. expected) number of all occurrences for each rule, we determine separate counts for each length (resp. interval) and use these to compute separate (expected) relative frequencies.

What remains to be shown is that the modified versions of the algorithms still provide rule probabilities that yield consistent grammars and maximize the likelihood.

Our proof in this section is based on the observation that an LSCFG is equivalent to an indexed grammar where each nonterminal symbol A is split into a set of indexed symbols A_i, and each rule $(A \to \alpha;\ p)$ is split into a set of rules $(A_i \to \beta;\ p(i) \cdot p_{\alpha,i})$, where β runs over all possible ways to index the nonterminal symbols in α such that the sum of the indices plus the number of terminal symbols in β is i. In this grammar, we can derive from each symbol A_i exactly those words that can be derived from A in the original grammar and have length i.

This indexed grammar is not context-free since it contains infinitely many nonterminal symbols and rules. It can be converted into an SCFG by removing all symbols and rules with indices larger than some threshold n. Obviously, this means we are losing the possibility to generate words longer than n. As we will see later on, however, our training algorithms will attribute a probability of zero to any word that is longer than the longest word in the training data. Thus, removing the possibility to generate such words will not influence the probability distribution.

Additionally, we introduce a new start symbol S' along with rules $(S' \to S_i;\ \frac{P_\ell(i)}{\sum_{0 \le j \le n} P_\ell(j)})$. The rescaling of probabilities ensures that they sum up to 1, even if our choice of n eliminated some words that had a nonzero probability.

For technical reasons, the following formal definition will introduce one additional modification: The rules $(A_i \to \beta;\ p(i) \cdot p_{\alpha,i})$ are split into two rules $(A_i \to A_{\alpha,i};\ p(i))$ and $(A_{\alpha,i} \to \beta;\ p_{\alpha,i})$ (where α is the unindexed counterpart of β). This reflects the distinction between choosing the symbols on the right-hand side and distributing the length among them, as introduced by our model. Furthermore, we include rules $(A_{\alpha,i} \to \beta;\ 0)$ for the cases where β can not be derived to a word of length i for technical reasons.

Definition 8.3. *For $G = (N, \Sigma, R, S, P_\ell)$ an* LSCFG *and $n \in \mathbb{N}$, the n-*indexed SCFG corresponding to *G is the* SCFG *$G_n = (N_n, \Sigma, R_n, S')$, where*

- $N_n = \{S'\} \sqcup \{A_i \mid A \in N, 0 \le i \le n\} \sqcup \{A_{\alpha,i} \mid (A \to \alpha;\ p) \in R,\ 0 \le i \le n\}$,
- $$\begin{aligned} R_n = {} & \{(S' \to S_i;\ \frac{P_\ell(i)}{\sum_{0 \le j \le n} P_\ell(j)}) \;\Big|\; 0 \le i \le n\} \\ & \cup \{(A_i \to A_{\alpha,i};\ p(i)) \;\Big|\; (A \to \alpha;\ p) \in R,\ 0 \le i \le n\} \\ & \cup \{(A_{\alpha,i} \to \beta;\ p') \;\Big|\; \beta = w^{(0)} A^{(1)}_{i_1} w^{(1)} \cdots w^{(k-1)} A^{(k)}_{i_k} w^{(k)},\ w^{(j)} \in \Sigma^*,\ 0 \le j \le k, \\ & \qquad (A \to \alpha;\ p) \in R,\ \alpha = w^{(0)} A^{(1)} w^{(1)} \cdots A^{(k)} w^{(k)},\ \sum_{j=0}^{k} (|w^{(j)}| + i_j) = i,\ i_0 = 0, \\ & \qquad p' = \begin{cases} p_{\alpha,i} & \exists w \in \Sigma^i :\ \beta \Rightarrow^*_{G'} w \\ 0 & \text{otherwise} \end{cases} \}, \end{aligned}$$

and G' denotes the CFG *that results from G by removing all probabilities.*

Lemma 8.4. *For an* LSCFG *G, let $w \in L(G)$, $|w| = m$, $n \ge m$. Then $w \in L(G_n)$, and there is a bijection between derivations $S \Rightarrow^*_G w$ and derivations $S_m \Rightarrow^*_{G_n} w$ such that corresponding derivations have the same probability.*
Additionally, $P_G(w) = P_{G_n}(w) \cdot \sum_{0 \le j \le n} P_{\ell,G}(j)$.

Proof. The first claim can be shown by structural induction on the derivations using the intuition given before Definition 8.3.

For the additional fact, we have

$$\begin{aligned} P_G(w) &= P_{\ell,G}(|w|) \cdot P_G(S \Rightarrow^* w) \\ &= P_{G_n}(S' \Rightarrow S_{|w|}) \cdot \sum_{0 \le j \le n} P_{\ell,G}(j) \ \cdot\ P_{G_n}(S_{|w|} \Rightarrow^* w) \\ &= P_{G_n}(w) \cdot \sum_{0 \le j \le n} P_{\ell,G}(j). \end{aligned}$$

□

Thus, the probability distribution on $L(G)$ induced by G_n converges to the one induced by G as $n \to \infty$. As mentioned before, our training algorithms will set $P_{\ell,G}(i) = 0$ if the training data contains no word of length i. Thus, by the above result training G is equivalent to training G_n, $n > \max_{t \in \mathcal{T}} |t|$, with the additional restrictions:

- For $(A_{\alpha,i} \to \beta;\ p) \in R_n$: $p \in \{p_{\alpha,i}, 0\}$.
- If i and j are in the same length interval and $(A_k \to A_{\alpha,k};\ p_k) \in R_n$, $k \in \{i, j\}$, then $p_i = p_j$.

The first condition expresses the fact that we have restricted ourselves to assume a uniform distribution of the probabilities among the various possibilities to distribute the length from the symbol on the left-hand side to those on the right-hand side. The second restriction expresses that the probability remains constant among all lengths in an interval. Obviously, this only applies if lengths are to be grouped into intervals. However, not all such groupings can yield a consistent grammar. As we will see in the following, one sufficient condition to guarantee this property is, to not group lengths i and j together whenever there is a rule that can lead to a word of length i but not to one of length j (or vice versa).

Definition 8.5. *Let $G = (N, \Sigma, R, S)$ a* CFG*, Q a partitioning of $\mathbb{N}$. We call Q* consistent with G *if it satisfies:*

$$\forall q \in Q,\ i, j \in q,\ (A \to \alpha) \in R\colon\ (\exists w_i \in \Sigma^i\colon\ \alpha \Rightarrow^* w_i \ \rightarrowtail\ \exists w_j \in \Sigma^j\colon\ \alpha \Rightarrow^* w_j).$$

Note that the partitioning into sets of one element each, which corresponds to not grouping lengths into intervals at all, is trivially consistent with each CFG G. Thus, this case is implicitly included in the following definitions and proofs.

Definition 8.6. *Let $G = (N, \Sigma, R, S)$ a* CFG*, Q a partitioning of $\mathbb{N}$ consistent with G, and $\mathcal{T}$ a set of full parse trees for G. For $r \in R,\ i \in \mathbb{N}$, we denote*

- *by $f(r, i, \mathcal{T})$ the number of applications of r in $\mathcal{T}$ with length i,*
- *by $f(r, \mathcal{T})$ the total number of applications of r in $\mathcal{T}$ and*
- *by $f(i, \mathcal{T})$ how many trees in $\mathcal{T}$ derive a word of length i.*

A function $P\colon\ R \to [0, 1]$ is the relative frequency estimator for G on $\mathcal{T}$ *iff* $\forall r = (A \to \alpha) \in R$:

$$P(r) = \begin{cases} \dfrac{f(r, \mathcal{T})}{\sum_{s \in R_A} f(s, \mathcal{T})} & \displaystyle\sum_{s \in R_A} f(s, \mathcal{T}) > 0, \\ \dfrac{1}{|R_A|} & \text{otherwise.} \end{cases}$$

A pair of functions $(P\colon\ R \times \mathbb{N} \to [0, 1],\ P_\ell\colon \mathbb{N} \to [0, 1])$ is the relative frequency estimator for G on $\mathcal{T}$ and Q *iff* $\forall r = (A \to \alpha) \in R,\ q \in Q,\ i \in q$:

$$P(r, i) = \begin{cases} \dfrac{\sum_{j \in q} f(r, j, \mathcal{T})}{\sum_{\substack{s \in R_A \\ j \in q}} f(s, j, \mathcal{T})} & \displaystyle\sum_{\substack{s \in R_A \\ j \in q}} f(s, j, \mathcal{T}) > 0, \\ \dfrac{1}{|R_A|} & \text{otherwise,} \end{cases}$$

and $\forall i \in \mathbb{N}$:

$$P_\ell(i) = \frac{f(i, \mathcal{T})}{|\mathcal{T}|}.$$

The estimates for rules with left-hand sides not appearing in $\mathcal{T}$ obviously do not affect the likelihood of $\mathcal{T}$. Thus, we are free to simply estimate uniform probabilities for these rules to ensure they add up to one.

The next step is to show that the length-dependent relative frequency estimator on G is consistent.

Lemma 8.7. *Let $G' = (N, \Sigma, R, S)$ a* CFG*, Q a partitioning of $\mathbb{N}$ consistent with G', $\mathcal{T}$ a set of full parse trees for G', (P, P_ℓ) the relative frequency estimator for G' on $\mathcal{T}$ and Q, and*

$$P(R) = \{(A \to \alpha;\ p) \,\big|\, r = (A \to \alpha) \in R \wedge \forall i \in \mathbb{N}\colon\ p(i) = P(r, i)\}.$$

Then $G = (N, \Sigma, P(R), S, P_\ell)$ is consistent.

Proof. For $n > \max_{t \in \mathcal{T}} |t|$ let $G_n = (N_n, \Sigma, R_n, S')$ the n-indexed SCFG corresponding to G. We show by induction on i that

$$\forall A \in N,\ i \in \mathbb{N}\colon\ a_i := \sum_{w \in \Sigma^*} P_n(A_i \Rightarrow^* w)\ \in \{0, 1\},$$

and if $a_i = 0$, then $\nexists w \in \Sigma^*\colon\ A_i \Rightarrow^* w$. From this, the result follows by Lemma 8.4.

$i = 0$: If there is no derivation $A \Rightarrow^*_{G'} \epsilon$, then $a_0 = 0$ by Definition 8.6.

For the other case let $\mathcal{T}'$ be the images of the parse trees in $\mathcal{T}$ under the bijection from Lemma 8.4, P'_n the relative frequency estimator for G_n on $\mathcal{T}'$, and finally

$$a'_0 = \sum_{A_0 \Rightarrow^* \epsilon} P'_n(A_0 \Rightarrow^* \epsilon).$$

Since G_n is a CFG, we can conclude from [CG98] that $\forall A \in N\colon\ a'_0 \in \{0, 1\}$ and if $a'_0 = 0$, then there is no derivation $A \Rightarrow^*_{G_n} \epsilon$.

By the definition of G_n, any derivation $A_0 \Rightarrow^*_{G_n} \epsilon$ can only consist of rules of the two forms $(A_0 \to A_{\alpha,0})$ and $(A_{\alpha,0} \to \beta)$. For the rules of the second type note that there is only one way to distribute length 0 among the symbols of α and thus

$$P'_n((A_{\alpha,0} \to \beta)) = \frac{1}{|(R_n)_{A_{\alpha,0}}|} = 1 = p_{\alpha,0} = P_n((A_{\alpha,0} \to \beta)).$$

For the first type, we have either

$$P'_n((A_0 \to A_{\alpha,0})) = \frac{1}{|R_{A_0}|} = P_n((A_0 \to A_{\alpha,0}))$$

or

$$\begin{aligned} P'_n((A_0 \to A_{\alpha,0})) &= \frac{f(r, \mathcal{T}')}{\sum_{s \in (R_n)_{A_0}} f(s, \mathcal{T}')} \\ &\overset{(1)}{=} \frac{f(r, 0, \mathcal{T}')}{\sum_{s \in (R_n)_{A_0}} f(s, 0, \mathcal{T}')} \\ &\overset{(2)}{=} P_G((A_0 \to A_{\alpha,0})) \\ &\overset{(3)}{=} P_n((A_0 \to A_{\alpha,0})), \end{aligned}$$

where (1) follows from the fact that A_0 can only be derived to words of length 0, (2) follows from the fact that the existence of a rule $(A \to \epsilon)$ by Definition 8.5 requires 0 to be in an interval of its own, and (3) follows from Definition 8.3.

Together, this gives $\forall A \in N\colon\ a_0 = a'_0$, proving the case $i = 0$.

$i \to i + 1$: For each $A \in N$, expanding a_i along all possibilities for the first two derivation steps yields the equation

$$a_i = \sum_{\substack{(A_i \to A_{\alpha,i};\ p) \in R_n \\ (A_{\alpha,i} \to \beta;\ p_{\alpha,i}) \in R_n \\ \beta \Rightarrow^* w}} p \cdot p_{\alpha,i} \cdot P_n(\beta \Rightarrow^* w).$$

For each of the possibilities for $\beta \in (N \cup \Sigma)^*$, the indices of the nonterminals plus the number of terminals in β add up to i. Thus, β contains either one nonterminal B_i along with zero or more nonterminals C_0 or an arbitrary number of terminals and nonterminals with smaller indices. In the first case, we get $\beta = \beta_1 B_i \beta_2$ and by the induction hypothesis

$$P_n(\beta \Rightarrow^* w) = P_n(\beta_1 \Rightarrow^* \epsilon) \cdot P_n(B_i \Rightarrow^* w) \cdot P_n(\beta_2 \Rightarrow^* \epsilon) \in \{0, b_i\}.$$

In the second case we get by the induction hypothesis $P_n(\beta \Rightarrow^* w) \in \{0, 1\}$. Thus,

$$P_n(\beta \Rightarrow^* w) \in \{b_i \mid B \in N\} \cup \{0, 1\}.$$

If $\sum_{\beta \Rightarrow^* w} P_n(\beta \Rightarrow^* w) = 0$, then at least one of the symbols in β contributes a factor of 0 and thus, by the induction hypothesis can not be derived to a word in Σ^*. Thus $\nexists w \in \Sigma^* \colon \beta \Rightarrow^* w$ and by Definition 8.3, $P_n((A_{\alpha,i} \to \beta) = 0$.

Using this and

$$\sum_{\substack{(A_i \to A_{\alpha,i};\, p) \in R_n,\\ (A_{\alpha,i} \to \beta;\, p_{\alpha,i}) \in R_n}} p \cdot p_{\alpha,i} = 1,$$

we may simplify each equation to either $a_i = 0$, $a_i = 1$ or

$$a_i = \sum_{B \in N} p_{i,AB} \cdot b_i + (1 - \sum_{B \in N} p_{i,AB}),$$

where $p_{i,AB} = \sum_{\substack{(A_i \to A_{\alpha,i};\, p) \in R_n,\\ (A_{\alpha,i} \to \beta;\, p_{\alpha,i}) \in R_n \colon \\ \beta = \beta_1 B_i \beta_2}} p \cdot p_{\alpha,i}$.

Now, let c_i and d_i denote the minimum and maximum respectively of the values a_i that have an equation of the third form. We find

$$\begin{aligned} c_i &= \sum_{B \in N} p_{i,CB} \cdot b_i + (1 - \sum_{B \in N} p_{i,CB}) \\ &\geq \sum_{B \in N} p_{i,CB} \cdot c_i + (1 - \sum_{B \in N} p_{i,CB}) \text{ and} \\ d_i &= \sum_{B \in N} p_{i,DB} \cdot b_i + (1 - \sum_{B \in N} p_{i,DB}) \\ &\leq \sum_{B \in N} p_{i,DB} \cdot d_i + (1 - \sum_{B \in N} p_{i,DB}), \end{aligned}$$

giving $c_i \geq 1$ and $d_i \leq 1$. Thus $a_i \in \{0, 1\}$ for any $A \in N$, and $a_i = 0$ only if each summand in the original equation is 0, implying $\nexists w \in \Sigma^* \colon A_i \Rightarrow^* w$.

□

Having established consistency, we only need to establish that the relative frequency estimator is a maximum-likelihood estimator.

Theorem 8.8. *Let $G = (N, \Sigma, R, S)$ a* CFG*, Q a partitioning of $\mathbb{N}$ consistent with G, $\mathcal{T}$ a set of full parse trees for G and (P, P_ℓ) the relative frequency estimator for G on $\mathcal{T}$ and Q. Then P is a maximum-likelihood estimator for length-dependent rule-probabilities on G and Q.*

Proof. Let P' be an arbitrary set of length-dependent probabilities on R. For each $A \in N$ and $q \in Q$, we define

$$P'_{A,q}((B \to \beta), i) = \begin{cases} P'((B \to \beta), i) & B = A \text{ and } i \in q \\ 1 & \text{else} \end{cases},$$

and for Δ a parse tree, we define $P'_\ell(\Delta) = P_\ell(|w_\Delta|)$, where w_Δ denotes the word generated by Δ. Then we find

$$L(\mathcal{T}, P') = L(\mathcal{T}, P'_\ell) \cdot \prod_{\substack{A \in N \\ q \in Q}} L(\mathcal{T}, P'_{A,q}) \cdot \prod_{\substack{r=(A \to \alpha) \in R \\ i \in \mathbb{N}}} (p_{\alpha,i})^{f(r,i,\mathcal{T})}. \tag{8.1}$$

On the right-hand side of (8.1), the rightmost product is independent of the probabilities and all the other factors can be maximized independent of each other. Since each of these factors is described by an unrestricted probability distribution on its defining domain, Theorem 1 from [Pre03] applies, stating that relative frequencies are a maximum-likelihood estimator for their probabilities.

Thus, by the definitions of P'_ℓ and of the $P'_{A,q}$ and by Equation (8.1), the claim follows. □

Theorem 8.9. *Let $G = (N, \Sigma, R, S)$ a CFG, Q a partitioning of $\mathbb{N}$ consistent with G and $\mathcal{T}$ a set of full parse trees for G. Then the inside-outside algorithm will converge to a set of rule probabilities that (locally or globally) maximizes the likelihood of observing $\mathcal{T}$.*

Proof. From Theorem 11 of [Pre03] and Theorem 8.8. □

8.4. Determining the Most Probable Derivation

In order to find the most probable derivation for a given primary structure, we can use modified versions of the parsing algorithms presented in Chapter 4 (cf. [Goo99] for the analogous modifications used to parse SCFG).

The central idea of the modification is that where the original parsers stored the information that $A \Rightarrow^* w_{i,j}$, we will now add to this the probability of the most probable derivation. Since all three parsers work bottom-up by combining known smaller partial derivations, we can always compute the probability of the newly found derivation as the product of the probabilities of its components.

For details, we will look at each parser individually.

8.4.1. CYK Algorithm

When parsing a word w of length n, the original CYK algorithm for FLEDCFG computes a table $\mathbf{T}$ such that for $0 \le i < j \le n$:

$$\mathbf{T}_{i,j} = \{A \in N \mid A \Rightarrow\!\![(i, j-i, n-j)]\!\!\Rightarrow^* w_{i+1\ldots j}\}.$$

The straightforward modification for LSCFG is to replace each nonterminal by a pair (nonterminal, probability). In order to increase readability we will use a slightly different approach and store a probability for each combination of nonterminal and subword, setting this value to 0 if the subword cannot be derived from the nonterminal in question.[3] This way for $0 \le i < j \le n$ and $A \in N$, we get:

$$\mathbf{T}_{i,j,A} = \max_\Delta \{P(A \Rightarrow\!\![\Delta]\!\!\Rightarrow w_{i+1\ldots j})\}.$$

For each possible entry $A \in \mathbf{T}_{i,i+l}$, the original algorithm checks possible derivations

$$\Delta = A \Rightarrow\!\![i, l, n-i-l]\!\!\Rightarrow BC \Rightarrow^* w_{i+1\ldots k} C \Rightarrow^* w_{i+1\ldots i+l},$$

where the existence of the second and third part can be read from the table. Now for LSCFG

$$P(A \Rightarrow\!\![\Delta]\!\!\Rightarrow w_{i\ldots i+l}) = p_r(l) \cdot P(B \Rightarrow\!\![\Delta_1]\!\!\Rightarrow w_{i+1\ldots k}) \cdot P(C \Rightarrow\!\![\Delta_2]\!\!\Rightarrow w_{k+1\ldots i+l}),$$

where $(A \to BC;\ p_r)$ is the rule applied in the first step and the probabilities for the second and third part can be taken from the table. Thus, the algorithm only needs to leave out the shortcut and store the highest probability found so far with each symbol. This results in:

[3] If necessary, this case can instead be handled by using a value outside of $[0, 1]$ and adding corresponding checks and special case handling to the algorithms.

for $i = 1 \ldots n$ **do**
  for each $A \in N$ **do**
    $\mathbf{T}_{i-1,i,A} := \max\{p(1) \mid (A \to w_i;\ p) \in R\}$;
    for $j = i+1 \ldots n$ **do**
      $\mathbf{T}_{i-1,j,A} := 0$;
for $l = 2 \ldots n$ **do**
  for $i = 0 \ldots n-l$ **do**
    for each $(A \to BC;\ p_r) \in R$ **do**
      for $k = i+1 \ldots i+l-1$ **do**
        $\mathbf{T}_{i,i+l} := \max\{\mathbf{T}_{i,i+l,A},\ p_r(l) \cdot p_B \cdot p_C\}$;

Adapting the backtracing algorithm so that it extracts a most probable leftmost derivation from the table is also straightforward:

parse(i,j,A)
if $j = i+1$ **then**
  print $(A \to w_i)$
else
  for each $r = (A \to BC) \in R$ **do**
    for $k = i+1 \ldots j-1$ **do**
      if $p_r(j-i) \cdot \mathbf{T}_{i,k,B} \cdot \mathbf{T}_{k,j,C} = \mathbf{T}_{i,j,A}$ **then**
        print $(A \to BC)$;
        parse(i,k,B);
        parse(k,j,C);
        return

8.4.2. Valiant's Algorithm

In Section 4.2.2, we used a specifically defined operation $*$ as multiplication and union as addition and continued these to a matrix product. For the modified matrices, we can use standard multiplication on the real numbers and the maximum operation as addition. This way, the matrix product becomes

$$(\mathbf{S} * \mathbf{T})_{i,j,A} = \max\{p_r(j-i) \cdot p_B \cdot p_C \mid (A \to BC;\ p_r) \in R,\ p_B = \mathbf{S}_{i,k,B},\ p_C = \mathbf{T}_{i,j,C},\ 0 \le k \le n\}.$$

Since max is commutative, the divide-and-conquer-strategy from Section 4.2.2 to compute $\mathbf{D}^+$ still applies. However, the reduction to the multiplication of binary matrices fails since the matrices S_A etc. now contain real numbers as entries. Since we also cannot use optimized library functions for real-valued matrix multiplication due to replacing addition with maximum, Valiant's algorithm is of even less interest in practice for [L]SCFG than for [FLED]CFG.

8.4.3. Earley's Algorithm

As stated in Section 4.2.3,

$$(A \to \alpha \bullet \beta) \in \mathbf{T}_{i,j} \quad \rightsquigarrow \quad \alpha \dashv[(i, j-i, n-j)]\models^* w_{i+1\ldots j} \\ \wedge\ \exists\delta\colon (S \Rightarrow^*_G \delta A\gamma \wedge \delta \dashv[(0, i, n-i)]\models^* w_{1\ldots i}).$$

For LSCFG this changes to

$$(A \to \alpha \bullet \beta;\ p) \in \mathbf{T}_{i,j} \quad \rightsquigarrow \quad \alpha \Rightarrow^* w_{i+1\ldots j} \\ \wedge\ \exists\delta\colon (S \Rightarrow^*_G \delta A\gamma \wedge \delta \Rightarrow^* w_{1\ldots i}) \\ \wedge\ p = \max\{P(\Delta) \mid \alpha \dashv[\Delta]\models w_{i+1\ldots j}\}.$$

The initialization then becomes $\mathbf{T}_{0,0} = \{(S \to \bullet\alpha;\ 1) \mid (S \to \alpha;\ p_r) \in R\}$, since the probability of the empty derivation is by definition the empty product and thus 1. For the three operations, we get:

Scanner: When an entry $(A \to \alpha \bullet a\beta;\ p) \in \mathbf{T}_{i,j}$ is present and $a = w_{j+1}$, add $(A \to \alpha a \bullet \beta;\ p)$ to $\mathbf{T}_{i,j+1}$.

Predictor: When an entry $(A \to \alpha \bullet B\beta;\ p) \in \mathbf{T}_{i,j}$, $B \in N$, is present, add $(B \to \bullet\gamma;\ 1)$ to $\mathbf{T}_{j,j}$ for each $(B \to \gamma;\ p_r) \in R$.

Completer: When entries $(A \to \alpha \bullet B\beta;\ p_1) \in \mathbf{T}_{i,k}$ and $(B \to \gamma\bullet;\ p_2) \in \mathbf{T}_{k,j}$ are present, and $(B \to \gamma;\ p_r) \in R$, add $(A \to \alpha B \bullet \beta;\ p = p_1 \cdot p_2 \cdot p_r(j-k))$ to $\mathbf{T}_{i,j}$ unless an entry $(A \to \alpha B \bullet \beta;\ p') \in \mathbf{T}_{i,j},\ p' > p$ is present. If such an entry with $p' < p$ is present, replace it.

To show the correctness of the modified algorithm we can carry over the proof from Section 4.2.3, just adding arguments concerning the probabilities:

Scanner: Since both items represent the same partial derivation they are annotated with the same probability.

Predictor: As for the initialization, the new item represents an empty derivation.

Completer: The new item represents the derivation $\alpha B \Rightarrow^* w_{i\ldots k}B \Rightarrow w_{i\ldots k}\gamma \Rightarrow^* w_{i\ldots j}$, and by the induction hypothesis the probabilities of the three parts are p_1, $p_r(j-i)$ and p_2, respectively. Thus, the probability of the complete derivation is $p_1 \cdot p_2 \cdot p_r(j-i)$. The clauses about items that only differ in the probability guarantee that only the one with the highest probability is kept in $\mathbf{T}_{i,j}$.

It should be noted that in Earley parsers for SCFG as found in the literature, the rules are typically initialized with p_r instead of 1 and that factor is then omitted from the completer. This does not work for LSCFG since the rule probability depends on the length of the generated subword, which is not known to the Predictor.

We conclude by giving the modified backtracing algorithm:

```
parse(i,j,(A → αC • β; p))
for each r = (C → γ) ∈ R do
  for k = i + 1 ... i + l − 1 do
    if (A → α • Cβ; p1) ∈ T_{i,k} ∧ (C → γ•; p2) ∈ T_{k,j} ∧ p_r(j − k) · p1 · p2 = p then
      if α = ϵ then
        print (A → αCβ);
      else
        parse(i,k,(A → α • Cβ));
      parse(k,j,(C → γ•));
      return
```

8.5. Experiments

In order to see if adding length-dependency actually improves the quality of predictions of RNA secondary structures from stochastic context-free grammars, we used length-dependent and traditional versions of four different grammars to predict the structures for two sets of RNA molecules for which the correct secondary structures were already known. Both sets were split into a training set which was used to train the grammars, and a benchmark set for which secondary structures were predicted using the trained grammars. We then compared these predicted structures to the structures from the database, computing two commonly used criteria to measure quality:

- **Sensitivity** (also called recall): The relative frequency of correctly predicted base pairs among base pairs that appear in the correct structure.

- **Specificity** (also called precision): The relative frequency of correctly predicted base pairs among base pairs that have been predicted.

Both frequencies were computed over the complete set (instead of calculating individual scores for each molecule and taking the average of these).

8.5.1. Data

In [DE04], Dowell and Eddy compared the prediction quality of several different grammars as well as some commonly used programs that predict RNA secondary structures by minimizing free energy. We decided to use the same data so our results are directly comparable to theirs.

Their training set consists of 139 each large and small subunit rRNAs, the benchmark dataset contains 225 RNase Ps, 81 SRPs and 97 tmRNAs. Both sets are available from the internet.[4] Since it contains different types of RNA, we will refer to this set as *mixed set* for the remainder of this chapter.

Additionally, we examined if length-dependent prediction is able to further improve the prediction quality for tRNA which is already predicted well by conventional SCFGs.

In order to do so, we took the tRNA database from [SVEB99], filtered out all sequences with unidentified bases, and split the remaining data into a training set of 1285 sequences and a benchmark set of 1284 sequences.

8.5.2. Grammars

We used 4 different grammars for our experiments:

G1:
$$S \to bS \mid aS\hat{a}S \mid \epsilon$$

G2:
$$\begin{aligned} S &\to LS \mid L \\ L &\to aF\hat{a} \mid b \\ F &\to aF\hat{a} \mid LS \end{aligned}$$

G3:
$$\begin{aligned} S &\to AS \mid bS \mid b \mid A \\ A &\to aA\hat{a} \mid aR\hat{a} \\ R &\to bT \mid AT \\ T &\to bT \mid AT \mid b \mid A \end{aligned}$$

G4:
$$\begin{aligned} S &\to CAS \mid C \\ C &\to bC \mid \epsilon \\ A &\to aL\hat{a} \\ L &\to aL\hat{a} \mid M \mid I \mid bH \mid \\ &\quad\ aL\hat{a}Bb \mid bBaL\hat{a} \\ B &\to bB \mid \epsilon \\ H &\to bH \mid \epsilon \\ I &\to bJaL\hat{a}Kb \\ J &\to bJ \mid \epsilon \\ K &\to bK \mid \epsilon \\ M &\to UaL\hat{a}UaL\hat{a}N \\ N &\to UaL\hat{a}N \mid U \\ U &\to bU \mid \epsilon \end{aligned}$$

In each of the grammars S is the start symbol, other capital letters denote nonterminal symbols, and $A \to \alpha \mid \beta$ is used as a shorthand for $A \to \alpha \in R$ and $A \to \beta \in R$.

As we have stated before, the grammars generate primary structures as words and encode secondary structures in the respective derivations. Thus, we have to distinguish (in the grammar) between terminal symbols representing unpaired bases and terminal symbols being part of a base pair. In the above grammars, the former are denoted by the symbol b while the latter are denoted by pairs $a, \hat{a}$.

Now, the straightforward approach of introducing a separate rule for each combination of how the symbols a, $\hat{a}$, and b can be replaced by the terminal symbols would – especially in the case of G4 – lead to a significant increase in the number of productions. To counter this, the authors of [DE04] translated the concept of transition and emission probabilities from hidden Markov models to SCFGs. The transition probability then is the probability of choosing, e.g., the rule scheme $(S \to bS)$ independent of which terminal symbol is used

[4] http://selab.janelia.org/software/conus/

in place of b. Then, the emission probability is the probability that a given b is a placeholder for a specific terminal symbol, independent of the rule which introduced the b.

In order to include this idea without having to extend the theory in Section 8.3, we left b in the grammar as a nonterminal and replaced all occurrences of $aL\hat{a}$ in G4 (or the corresponding substrings in the other grammars) with A, adding rules that generate all (combinations of) terminal symbols a, c, g and u from these nonterminals.

G1 and G2 have been taken from [DE04], G1 being the simplest grammar in the comparison and G2, which originates from [KH99], being the grammar which achieved the best results. G2 and G3 both extend G1 based on the observation that a secondary structure will be more stable if it contains longer runs of immediately nested base pairs. They differ in the approach taken to include this into the model.

G4 has been taken from [Neb04]. It models the decomposition that is used for minimum free energy prediction. Since a run of unpaired bases contributes differently to the free energy depending on the context it appears in, these runs are derived from a separate nonterminal symbol (B, C, H, I, J, or U), depending on the context with the intention of capturing these different contributions in the rule probabilities.

As we stated in Section 8.2, we implemented length-dependency such that we grouped the lengths into intervals, the rule probabilities changing only from one interval to the other but not within them.

We decided to increase the size of the intervals as the considered subwords get longer, since the influence a change in length has on the probabilities most likely depends on relative changes rather than absolute ones. This also helps to keep the estimated probabilities accurate since any training set will naturally contain fewer data points per length as length increases.

Aside from this consideration and the restrictions implied by Definition 8.5 (consistency of a set of intervals), there is no obvious criterion that helps deciding on a set of intervals.

Thus, we created several different sets ranging from ≈ 10 to ≈ 100 intervals, evaluating a subset of the prediction data with each of them. The results corresponded to our expectation that finer intervals tend to improve the quality of prediction until, at some point, the amount of data available to estimate each probability is too sparse and thus the quality degrades. The following set of intervals yielded the best or close to the best results for all 4 grammars:

Lengths up to 40 were not grouped at all (i.e. for each of these lengths an individual set of probabilities was computed), lengths from 41 to 60 were grouped in pairs, then sets of three lengths up to 84, sets of four up to 100, sets of five up to 140, sets of ten up to 200, twenty lengths per set up to 300, $[301;330]$, $[331;360]$, $[361;390]$, $[391;430]$, $[431;470]$, $[471;510]$, and $[511;\infty]$. Since all structures in the benchmark sets are shorter than 500 bases, the probabilities of the last interval did not influence predictions.

At a first glance it may seem surprising that on G4 the same number of intervals can be used as on the smaller grammars given the greater number of rules and thus greater number of probabilities that have to be trained. However, a closer look shows that for all nonterminal symbols in G4 except for S, L and N, there is at most one applicable rule for each length, leaving only 10 transition probabilities per interval to be actually determined by training, which is the same number as for G3 and already less than the number of emission probabilities.

8.5.3. Observations and Dicussion

We trained and predicted using a length-dependent version of the Earley-style-parser from [Wil10]. The results of the benchmarks are listed in Table 8.1.

Looking at these results, it is immediately apparent that the predictions from G1 are too bad to be of any use both with or without lengths. Additionally, the higher complexity of G4 compared to G2 and G3 did not lead to an improvement in prediction quality.

Concerning the other grammars, we note that adding length-dependency significantly improved the results on tRNA while they became worse on the mixed set.

A possible explanation for these results could be that the correct parameters for folding are different for different types of RNA. In order to test this hypothesis, we took the three types of RNA in the mixed benchmark set, splitting the molecules of each type into a training set containing 2/3 of them, and a benchmark set

Table 8.1.: Grammar performance, given as "sensitivity/specificity", values rounded to full percent.

Grammar	Mixed Set		tRNA Set	
	without lengths	with lengths	without lengths	with lengths
G1	2 / 2	2 / 3	6 / 6	6 / 20
G2	48 / 45	35 / 27	80 / 80	96 / 79
G3	40 / 48	40 / 41	78 / 81	95 / 96
G4	39 / 47	22 / 54	78 / 83	84 / 95

Table 8.2.: Grammar performance, given as "sensitivity/specificity", values rounded to full percent.

Grammar	RNaseP Set		SRP Set		tmRNA Set	
	without lengths	with lengths	without lengths	with lengths	without lengths	with lengths
G2	50 / 47	46 / 36	57 / 52	42 / 34	39 / 36	41 / 23
G3	47 / 45	52 / 53	57 / 52	59 / 54	38 / 36	45 / 59

containing the remaining molecules. On these three sets we again trained and predicted the grammars G2 and G3, them being the most likely candidates for future use. The results are listed in Table 8.2.

For each of the sets G3 with lengths performed best, backing our assumption. Additionally, while the non-length-dependent versions of the grammar performed almost equally, the length-dependent version of G2 fell behind G3 significantly, indicating that length-dependent prediction is more sensitive to the choice of grammar.

8.5.4. Runtime

The considerations in Chapter 4 and Section 8.4 lead to the assumption that both versions should take about the same time on a given grammar. This was confirmed during our experiments. None of the versions was consistently faster, i.e. if there is a difference, it was overshadowed by effects like system load.

Concerning the different grammars, predictions using G1 were faster than those for G2 by a factor of ≈ 1.5. Between G2 and G3 resp. G3 and G4 the factor was ≈ 2. These factors match the number of rules in the different grammars.

8.5.5. Second Experiment

In [SN12] and Chapter 7 of [Sch12], Schulz and Nebel present the results of another experiment.

Differing from our approach, they did not compute the most probable derivation but instead drew a random sample of 1000 structures according to the distribution implied by the grammar. From this sample, they computed their prediction using several approaches (most common structure in the sample, structure with highest probability, consensus).

Independent of the method used to predict structures, their observations confirmed what we have seen above: lSCFG outperform SCFG when trained on RNA of a specific type but they perform worse on mixed sets of RNA.

9. Conclusion and Outlook

In this thesis, we have introduced a new type of restricted rewriting. We allow or disallow rule applications in a context-free base grammar based on the length of the subword that results from the application, resp. its position in the complete word.

We have defined several subclasses by allowing only a subset of the types of restrictions, requiring different restrictions on a single rule to be independent, or using right or left linear grammars as a basis. We have then shown that these subclasses form a hierarchy of distinct languages above the context-free resp. regular languages.

Furthermore, given a reasonable condition on the restriction sets, namely that membership for them is decidable in linear time, our grammars can be parsed in the same asymptotic time as traditional context-free grammars.

We have also shown that, aside of membership, none of the commonly studied decidability problems are decidable for our new grammar classes. While some closure properties carry over from context-free resp. regular languages, others do not.

Additionally, we have given an equivalent class of automata for each of the grammar classes.

Finally, we have defined length-dependent stochastic context-free grammars and we have shown that they can replace traditional stochastic context-free grammars in algorithms that predict the secondary structure of RNA molecules. We have demonstrated that LSCFG improve the quality of predictions when trained and used on a set consisting of RNA of a single type, while they do not improve the quality of predictions on mixed sets of RNA.

9.1. Possible Future Work

There are several directions in which the research presented in this thesis can be continued. One obvious way would be to consider other types of base grammars beyond the context-free and right/left linear grammars examined here.

9.1.1. Other base grammars

The first options that come to mind are other classes from the Chomsky hierarchy, i.e. context-sensitive and type-0 grammars. For type-0 grammars, we run into the problem that it is unclear how the yield of a rule application should be defined. If several nonterminals that were introduced in separate rule applications are replaced at once, there is no indication how the yield of the newly introduced symbols should be distributed among the replaced symbols.

For context-sensitive grammars, the situation is easier, especially when viewing the rules from a restricted rewriting point of view as context-free rules which may only be applied in a given local context. It then becomes natural to simply ignore the unmodified context when defining length and position of a rule application.

Similar to context-free-based grammars, we can describe any context-sensitive language with these grammars, and as long as the plsets are decidable in linear space, the grammars will not describe non-context-sensitive languages. For more complex plsets, we also expect results analogous to FLEDCFG.

Another option is to consider linear base grammars. Since REG $\subsetneq$ LIN $\subsetneq$ CFL, we expect a hierarchy between the context-free-based and regular-based language classes. A closer look at the results from Chapter 5.1 supports this hypothesis.

Since the Dyck language $L_D \notin$ LIN, and since we have seen that position and length dependencies can not be used to ensure balancedness of parentheses, we can conclude that this language will separate the

linear-based from the context-free-based classes. To separate the regular-based and linear-based classes, a simple application of Theorem 5.3 shows that $\{ww^R\} \notin$ fedREG.

Furthermore, the automaton from Lemma 5.6 corresponds to a linear grammar since it only adds symbols to the stack during the first half of the computation, and only removes them during the second half of the computation. Also, the separating language in Lemma 5.12 is linear.

Thus, the only piece missing to completely prove that the linear-based language classes are parallel to the context-free-based classes is a language in lLIN \ fedCFL. It seems likely that such a language can be derived from the one given in Lemma 5.11 with some effort.

Other base grammar classes that might be interesting are deterministic and unambiguous context-free grammars. Just as the linear-based grammars, we expect them to form separate hierarchies between the regular-based and context-free-based classes.

Another option is to combine position and length-dependencies with other modes of restricted rewriting. As we have seen for context-sensitive base grammars there is not much to be gained if the class without position-and-length-dependencies is already rather large, but valence grammars and multiple context-free grammars might be worth looking at.

9.1.2. Extending the Application

Aside from considerations in formal language theory, the application of lSCFG also offers several possibilities for future research.

The first thing that comes to mind is to not only use length-dependencies but also include positions in the predictions. However, at a closer look this seems less promising. Including positions in the form of k position intervals would mean that the number of probabilities that have to be trained for each rule increases k-fold. Thus, the required amount of training data increases accordingly. Given that the length-dependent model only yields good prediction results when trained on a set of RNA of a single type, the same should be expected for the position-and-length-dependent model. Thus, we would need very large sets of reliable secondary structures of a single type of RNA to train the model. Unfortunately, such sets do not exist ([RLE12]).

Another potential area of research is extending other grammar-based approaches of structure prediction by length-dependencies.

Stochastic MCFG have already been used to model and predict secondary structures with pseudoknots ([KSK06], [NW12]) or of two molecules folding together ([KAS09]). However, this approach appears to be even more limited by a lack of available training data than the one presented before.

Recently, zu Siederdissen et. al. introduced a model that incorporates the fact that each base actually has three sites that allow for interaction and they may all be active at the same time ([zSBSH11]). This model can also be translated into a context-free grammar ([Mü13], [MN]), and thus length-dependencies can be added. As for the previous approaches, success depends on the availability of suitable training sets.

Finally, we want to mention the possibility of using lSCFG in other fields where SCFG are already used like natural language processing.

Bibliography

[Abr65] Samuel Abraham. Some questions of phrase structure grammars. *Computational Linguistics*, 4: 61–70, 1965.

[And10] Ebbe Sloth Andersen. Prediction and design of DNA and RNA structures. *New Biotechnology*, 27(3):184 – 193, 2010.

[AU72] Alfred V Aho and Jeffrey D Ullman. *The Theory of Parsing, Translation, and Compiling: Volume I: Parsing*. Prentice-Hall, Incorporated, 1972.

[BF94] Henning Bordihn and Henning Fernau. Accepting grammars with regulation. *International Journal of Computer Mathematics*, 53(1-2):1–18, 1994.

[BMCW81] Walter Bucher, Hermann A. Maurer, Karel Culik, and Detlef Wotschke. Concise description of finite languages. *Theoretical Computer Science*, 14(3):227–246, 1981.

[BRK80] John Boyle, George T. Robillard, and Sung-Hou Kim. Sequential folding of transfer RNA: A nuclear magnetic resonance study of successively longer tRNA fragments with a common 5′ end. *Journal of Molecular Biology*, 139(4):601–625, 1980.

[CG98] Zhiyi Chi and Stuart Geman. Estimation of probabilistic context-free grammars. *Computational Linguistics*, 24(2):299–305, 1998.

[Cho59] Noam Chomsky. On certain formal properties of grammars. *Information and control*, 2(2): 137–167, 1959.

[CM73] Armin B. Cremers and Otto Mayer. On matrix languages. *Information and Control*, 23(1):86–96, 1973.

[CM74] Armin B. Cremers and Otto Mayer. On vector languages. *Journal of Computer and System Sciences*, 8(2):158–166, 1974.

[Cre73] Armin B. Cremers. Normal forms for context-sensitive grammars. *Acta Informatica*, 3(1):59–73, 1973.

[DE04] Robin D. Dowell and Sean R. Eddy. Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC Bioinformatics*, 5:71, 2004.

[DM12] Jürgen Dassow and Tomáš Masopust. On restricted context-free grammars. *Journal of Computer and System Sciences*, 78(1):293–304, 2012.

[DP89] Jürgen Dassow and Gheorghe Paun. Regulated rewriting in formal language theory, volume 18 of EATCS monographs in theoretical computer science. *Springer-Verlag*, 1989.

[DPS97] Jügen Dassow, Gheorghe Păun, and Arto Salomaa. Grammars with controlled derivations. In *Handbook of formal languages, vol. 2*, pages 101–154. Springer-Verlag New York, Inc., 1997.

[DT09] Jürgen Dassow and Sherzod Turaev. Petri net controlled grammars: the power of labeling and final markings. *Romanian Journal of Information Science and Technology*, 12(2):191–207, 2009.

[Ear70] Jay Earley. An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, 13:94–102, 1970.

[Fer96] Henning Fernau. On grammar and language families. *Fundamenta Informaticae*, 25(1):17–34, 1996.

[Flo62] Robert W. Floyd. On the nonexistence of a phrase structure grammar for ALGOL 60. *Communications of the ACM*, 5(9):483–484, 1962.

[Fri68] Ivan Frič. Grammars with partial ordering of the rules. *Information and Control*, 12(5):415 – 425, 1968.

[FS02] Henning Fernau and Ralf Stiebe. Sequential grammars and automata with valences. *Theoretical Computer Science*, 276(1):377–405, 2002.

[GH69] Sheila Greibach and John Hopcroft. Scattered context grammars. *Journal of Computer and System Sciences*, 3(3):233–247, 1969.

[GHR80] Susan L. Graham, Michael A. Harrison, and Walter L. Ruzzo. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2:415–462, 1980.

[GHR92] Yonggang Guan, Günter Hotz, and A. Reichert. Tree grammars with multilinear interpretation. Technical report, Universitäts- und Landesbibliothek Saarbrücken, 1992.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and intractability*, volume 174. freeman New York, 1979.

[Goo99] Joshua Goodman. Semiring parsing. *Computational Linguistics*, 25(4):573–605, 1999.

[Gre67] Ulf Grenander. *Syntax-controlled probabilities*. Division of Applied Mathematics, Brown University, 1967.

[GRS80] Ronald L. Graham, Bruce L. Rothschild, and Joel H. Spencer. *Ramsey theory*. Wiley New York, 1980.

[GS68] Seymour Ginsburg and Edwin H. Spanier. Control sets on grammars. *Mathematical Systems Theory*, 2(2):159–177, 1968.

[Gua92] Yonggang Guan. *Klammergrammatiken, Netzgrammatiken und Interpretationen von Netzen*. PhD thesis, Universität des Saarlandes, 1992.

[GW89] Jakob Gonczarowski and Manfred K. Warmuth. Scattered versus context-sensitive rewriting. *Acta Informatica*, 27(1):81–95, 1989.

[Har78] Michael A. Harrison. *Introduction to Formal Language Theory*. Addison Wesley, 1978.

[HJ94] Dirk Hauschildt and Matthias Jantzen. Petri net algorithms in the theory of matrix grammars. *Acta Informatica*, 31(8):719–728, 1994.

[HP96] Günter Hotz and Gisela Pitsch. On parsing coupled-context-free languages. *Theoretical Computer Science*, 161(1):205–233, 1996.

[HU79] John E Hopcroft and Jeffrey D Ullman. Introduction to automata theory, languages, and computation, 1979.

[Jos85] Aravind Krishna Joshi. *Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions?* University of Pennsylvania, Moore School of Electrical Engineering, Department of Computer and Information Science, 1985.

[Kal10] Laura Kallmeyer. *Parsing beyond context-free grammars*. Springer, 2010.

[Kas65] Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA, 1965.

[KAS09] Yuki Kato, Tatsuya Akutsu, and Hiroyuki Seki. A grammatical approach to rna–rna interaction prediction. *Pattern Recognition*, 42(4):531–538, 2009.

[Kel84] Jozef Kelemen. Conditional grammars: Motivations, definitions, and some properties. In *Proc. Conf. Automata, Languages and Mathematical Sciences*, pages 110–123, 1984.

[KH99] Bjarne Knudsen and Jotun Hein. RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. *Bioinformatics*, 15(6):446–454, 1999.

[KSF88] Tadao Kasami, Hiroyuki Seki, and Mamoru Fujii. Generalized context-free grammars, multiple context-free grammars and head grammars. *Preprint of WG on Natural Language of IPSJ*, 1988.

[KSK06] Yuki Kato, Hiroyuki Seki, and Tadao Kasami. RNA pseudoknotted structure prediction using stochastic multiple context-free grammar. *IPSJ Digital Courier*, 2:655–664, 2006.

[Mas09] Tomáš Masopust. A note on the generative power of some simple variants of context-free grammars regulated by context conditions. In *Language and Automata Theory and Applications*, pages 554–565. Springer, 2009.

[May72] Otto Mayer. Some restrictive devices for context-free grammars. *Information and Control*, 20(1): 69–92, 1972.

[Mic05] Jens Michaelis. An additional observation on strict derivational minimalism. *Proceedings of FG-MoL*, pages 101–111, 2005.

[MM04] Irmtraud Meyer and Istvan Miklos. Co-transcriptional folding is encoded within RNA genes. *BMC Molecular Biology*, 5(1):10, 2004.

[MN] Robert Müller and Markus E. Nebel. Combinatorics of RNA secondary structures with base triples. submitted.

[MR71] David L. Milgram and Azriel Rosenfeld. A note on scattered context grammars. *Information Processing Letters*, 1(2):47–50, 1971.

[Mü13] Robert Müller. Combinatorics of RNA secondary structures with base triples. Bachelor's Thesis, TU Kaiserslautern, 2013.

[Neb04] Markus E. Nebel. Identifying good predictions of RNA secondary structure. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 423–434, 2004.

[Neb12] Markus Nebel. *Formale Grundlagen der Programmierung*. Springer, 2012.

[Ner58] Anil Nerode. Linear automaton transformations. *Proceedings of the AMS*, 9:541–544, 1958.

[NPGK78] Ruth Nussinov, George Pieczenik, Jerrold R. Griggs, and Daniel J. Kleitman. Algorithms for loop matchings. *SIAM Journal on Applied mathematics*, 35(1):68–82, 1978.

[NW12] Markus E. Nebel and Frank Weinberg. Algebraic and combinatorial properties of common RNA pseudoknot classes with applications. *Journal of Computational Biology*, 19(10):1134–1150, 2012.

[ORW85] William Ogden, Rockford J. Ross, and Karl Winklmann. An "Interchange Lemma" for context-free languages. *SIAM Journal on Computing*, 14(2):410–415, 1985.

[Pău79] Gheorghe Păun. On the generative capacity of conditional grammars. *Information and Control*, 43(2):178–186, 1979.

[Pau80] Gheorghe Paun. A new generative device: valence grammars. *Rev. Roumaine Math. Pures Appl*, 25(6):911–924, 1980.

[Pit93] Gisela Pitsch. *Analyse von Klammergrammatiken*. PhD thesis, Technische Fakultät der Universität des Saarlandes, 1993.

[Pos46] Emil Leon Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52:122–134, 1946.

[Pre03] Detlef Prescher. A tutorial on the expectation-maximization algorithm including maximum-likelihood estimation and em training of probabilistic context-free grammars. http://arxiv.org/pdf/cs/0412015, 2003.

[Păb85] Gheorghe Păun. A variant of random context grammars: Semi-conditional grammars. *Theoretical Computer Science*, 41:1–17, 1985.

[RLE12] Elena Rivas, Raymond Lang, and Sean R. Eddy. A range of complex probabilistic models for RNA secondary structure prediction that includes the nearest-neighbor model and more. *RNA*, 18 (2):193–212, 2012.

[Ros69] Daniel J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the ACM (JACM)*, 16(1):107–131, 1969.

[RS78] Imre Z Ruzsa and Endre Szemerédi. Triple systems with no six points carrying three triangles. *Combinatorics (Keszthely, 1976), Coll. Math. Soc. J. Bolyai*, 18:939–945, 1978.

[Sal69] Arto Salomaa. *On grammars with restricted use of productions*. Suomalainen tiedeakatemia, 1969.

[Sal70] Arto Salomaa. Periodically time-variant context-free grammars. *Information and control*, 17(3): 294–311, 1970.

[Sat96] Giorgio Satta. The membership problem for unordered vector grammars. *Developments in Language Theory II*, pages 267–275, 1996.

[Sch12] Anika Schulz. *Sampling and Approximation in the Context of RNA Secondary Structure Prediction*. PhD thesis, Technische Universität Kaiserslautern, 2012.

[Sea92] David B. Searls. The linguistics of DNA. *American Scientist*, 80(6):579–591, 1992.

[Shi87] Stuart M. Shieber. Evidence against the context-freeness of natural language. In *The Formal complexity of natural language*, pages 320–334. Springer, 1987.

[SMFK91] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229, 1991.

[SN12] Anika Scheid and Markus E. Nebel. Statistical RNA secondary structure sampling based on a length-dependent SCFG model. Technical report, Technische Universität Kaiserslautern, 2012.

[Sto94] Andreas Stolcke. *Bayesian Learning of Probabilistic Language Models*. PhD thesis, University of California at Berkeley, 1994.

[SVEB99] Mathias Sprinzl, Konstantin S. Vassilenko, J. Emmerich, and F. Bauer. Compilation of tRNA sequences and sequences of tRNA genes. http://www.uni-bayreuth.de/departments/biochemie/trna/, 20 December 1999.

[Val75] Leslie G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308–314, 1975.

[VdW70] APJ Van der Walt. Random context grammars. In *Proceedings of the Symposium on Formal Languages*, pages 163–165, 1970.

[vdWE00] Andries van der Walt and Sigrid Ewert. A shrinking lemma for random forbidding context languages. *Theoretical Computer Science*, 237(1):149–158, 2000.

[Wil10] Sebastian Wild. An Earley-style parser for solving the RNA-RNA interaction problem. Bachelor's Thesis, TU Kaiserslautern, 2010. https://kluedo.ub.uni-kl.de/frontdoor/index/index/docId/2285.

[Wil11] Virginia Williams. Breaking the Coppersmith-Winograd barrier. Manuscript, 2011.

[WZ] Frank Weinberg and Georg Zetzsche. Some results on vector grammars with appearance checking. unpublished.

[XBI05] A. Xayaphoummine, T. Bucher, and H. Isambert. Kinefold web server for RNA/DNA folding path and structure prediction including pseudoknots and knots. *Nucleic acids research*, 33(suppl 2):W605–W610, 2005.

[Yam08] Tomoyuki Yamakami. Swapping lemmas for regular and context-free languages. *arXiv preprint arXiv:0808.4122*, 2008.

[You67] Daniel H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.

[Zet10] Georg Zetzsche. On erasing productions in random context grammars. In *Automata, Languages and Programming*, volume 6199, pages 175–186. Springer Berlin Heidelberg, 2010.

[Zet11a] Georg Zetzsche. A sufficient condition for erasing productions to be avoidable. In *Developments in Language Theory*, pages 452–463. Springer, 2011.

[Zet11b] Georg Zetzsche. Toward understanding the generative capacity of erasing rules in matrix grammars. *International Journal of Foundations of Computer Science*, 22(02):411–426, 2011.

[ZS81] Michael Zuker and Patrick Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic acids research*, 9(1):133–148, 1981.

[zSBSH11] Christian Höner zu Siederdissen, Stephan H. Bernhart, Peter F. Stadler, and Ivo L. Hofacker. A folding algorithm for extended RNA secondary structures. *Bioinformatics*, 27(13):i129–i136, 2011.

A. Index of Notations

Notation	Meaning	Introduced in
Variables and Symbols		
$a, b, c, \ldots$	Terminal symbol	
f, l, e	f-distance, length, e-distance	Section 1.5
g	Single stack symbol	
g_0	Stack start symbol	
$i, j, k, m, n, \ldots$	Integer	
p	Rule probability	
p_r	Rule probability of rule r	
r	Grammar rule	
$u, v, w, \ldots$	String of terminal symbols	
$A, B, C, \ldots$	Nonterminal symbol or Automaton	
F	Final states of an automaton	
G	Grammar	
L	Language	
M	plset	
N	Nonterminal alphabet	
P	Probability function	
Q	States of an automaton	
R	Rules of a grammar	
R_A	Rules with left-hand side A	
S	Start symbol of a grammar	
$S, T, U, \ldots$	Nonterminal symbols	
$\mathbf{T}$	Table of items of a parser	Section 4.2
$\mathcal{T}$	Set of parse trees	

Variables and Symbols (continued)

$\alpha, \beta, \ldots$	String of terminal and nonterminal symbols	
γ	String of stack symbols	
δ	Transition function	
ϵ	Empty word	Section 1.2
Γ	Stack alphabet	
Σ	Terminal alphabet	
\$	Bottom of stack	
$\rightsquigarrow$	Implication	Section 1.2
$\leftrightsquigarrow$	Equivalence	Section 1.2

Sets

$\mathbb{N}$	The natural numbers including 0	
$\mathbb{N}_+$	The natural numbers without 0	
$\wp M$	Power set of M	
$M_1 \times \cdots \times M_k$	$\{(m_1, \ldots, m_k) \mid m_i \in M_i, 1 \leq i \leq k\}$	
M^i	$\underbrace{M \times \cdots \times M}_{i \text{ factors}}$	
$M_1 + M_2$	$\{m_1 + m_2 \mid m_1 \in M_1, m_2 \in M_2\}$	Section 1.2
$M_1 - M_2$	$\{m_1 - m_2 \mid m_1 \in M_1, m_2 \in M_2\}$	Section 1.2
$M_1 \cdot M_2$	$\{m_1 m_2 \mid m_1 \in M_1, m_2 \in M_2\}$	Section 1.2
$M_1^{M_2}$	$\{m_1^{m_2} \mid m_1 \in M_1, m_2 \in M_2\}$	Section 1.2
$M_1 \sqcup M_2$	Disjoint union of M_1 and M_2	Section 1.2
$M/\sim$	Equivalence classes of M under the relation $\sim$	Theorem 5.2

Strings and Languages

$\lvert w \rvert$	Length of w	
$\lvert w \rvert_a$	number of a's in w	
w_i	i-th character of w (counting starts at 1)	Section 1.2
$w_{i\ldots j}$	$w_i \cdots w_j$	Section 1.2
w^R	Reversal of w	Section 1.2
$v \cdot w$	Concatenation vw	Section 1.2

Strings and Languages (continued)

Σ_ϵ	$\Sigma \cup \{\epsilon\}$	Section 1.2
Σ^*	Words over Σ	
Σ^+	Nonempty words over Σ	
Σ^i	Words of length i over Σ	
Σ^M	$\bigcup_{i \in M} \Sigma^i$	
a^*, a^+, a^i, a^M	$\{a\}^*, \{a\}^+, \{a\}^i, \{a\}^M$	Section 1.5
L^R	$\{w^R \mid w \in L\}$	Lemma 2.12
$L(G)$	Language generated by G	Definition 2.4
$T(A)$	Language accepted by A by final state	Definition 3.3
$N(A)$	Language accepted by A by empty stack	Definition 3.11
$L(A)$	Language accepted by A by final state and empty stack	Definition 3.11

Derivations and Computations

$(A \to \alpha)$	Unrestricted grammar rule	Section 1.5
$(A \to \alpha;\ M)$	Grammar rule restricted by M (M can be a plset, permitted or forbidden context, monoid element, etc. depending on the type of grammar.)	Section 1.5
$\text{yield}(\alpha)$	Subword of the final word that is eventually derived from α	Section 1.5
$\alpha \Rightarrow \beta$	Derivation from α to β in a single step	Definition 2.3
$\alpha \Rightarrow^k \beta$	Derivation in k steps	Definition 2.3
$\alpha \Rightarrow^* \beta$	Derivation in any number of steps	Definition 2.3
$\alpha \Rightarrow[r_1, \ldots, r_k] \beta$	Derivation using rules $r_1, \ldots, r_k$ in this order	Definition 2.3
$\alpha \Rightarrow[; L]^* \beta$	Leftmost derivation	Definition 2.3
$\alpha \Rightarrow[(f,l,e)]^* \beta$	Derivation assuming $\lvert\text{yield}(\beta)\rvert = l$ and a context of f characters to the left and e characters to the right	Definition 2.3
$\alpha \Rightarrow\left[\begin{smallmatrix} r_1,\ldots,r_k \\ (f,l,e) \end{smallmatrix}\right] \beta$	Derivation using rules $r_1, \ldots, r_k$ at postition and length (f, l, e)	Definition 2.2
$\alpha \Rightarrow_G^* \beta$	Derivation in the grammar G	Definition 2.3
$uqvw \vdash uvq'w$	Single-step computation consuming v and changing state from q to q'	Definition 3.3
$uqvw \vdash^k uvq'w$	k-step computation	Definition 3.3
$uqvw \vdash^* uvq'w$	Computation with an arbitrary number of steps	Definition 3.3
$uqvw \vdash_A^* uvq'w$	Computation of the automaton A	Definition 3.3

Grammar, Language and Automata Classes

ωG	Grammar class	Section 1.2
ωL	Set of languages generated by grammars in ωG	Section 1.2
[x]C	The statement holds for xC and C	Section 1.2
[x]C = [x]D	xC = xD and C = D	Section 1.2
RE	Recursive enumerable languages	Section 1.2
DEC	Decidable languages	Section 1.2
CSG	Context-sensitive grammars	Section 1.2
CFG	Context-free grammars	Section 1.2
RLING	Right-linear grammars	Section 1.2
LLING	Left-linear grammars	Section 1.2
DFA	Deterministic finite automata	Definition 3.8
NFA	Nondeterministic finite automata	Definition 3.1
PDA	Pushdown automata	Definition 3.10
[F][L][E][D]CFG	Position-and-length-dependent context-free grammars	Section 1.5
[F][L][E][D]RLING	Position-and-length-dependent right-linear grammars	Section 1.5
[F][L][E][D]LLING	Position-and-length-dependent left-linear grammars	Section 1.5
[F][E][D]REG	Regular-based languages	Definition 3.4
[F][E][D]DFA	Deterministic finite automata with position counter	Definition 3.8
[F][E][D]NFA	Nondeterministic finite automata with position counter	Definition 3.1
[F][L][E][D]PDA	Pushdown automata with position counter	Definition 3.10
Prefix F	Restrictions based on f-distance are allowed	Section 1.5
Prefix L	Restrictions based on length are allowed	Section 1.5
Prefix E	Restrictions based on e-distance are allowed	Section 1.5
Prefix D	Restrictions may be mutually dependent	Section 1.5
[ϵ]CG	Conditional grammars	Section 1.4.1
CCFG	Coupled context-free grammars	Section 1.4.3
[ϵ][A][U]MG	Matrix grammars	Section 1.4.2
MCFG	Multiple context-free grammars	Section 1.4.3
[ϵ][A]PG	Programmed grammars	Section 1.4.2

Grammar, Language and Automata Classes (continued)

[ϵ][A]PNG	Petri net grammars	Section 1.4.2
[ϵ][A]RCG	Regularly controlled grammars	Section 1.4.2
[ϵ]RCG	Random context grammars	Section 1.4.1
[ϵ]FRCG	Forbidding random context grammars	Section 1.4.1
[ϵ]PRCG	Permitting random context grammars	Section 1.4.1
[ϵ]sCG	Semi-conditional grammars	Section 1.4.1
[ϵ][A][U]SCG	Scattered context grammars	Section 1.4.2
[ϵ][A][U]VG	Vector grammars	Section 1.4.2
$\mathbb{Q}^i$VG	Additive valence grammars	Section 1.4.2
$\mathbb{Z}^i$VG	Multiplicative valence grammars	Section 1.4.2
Prefix ϵ	ϵ-rules are allowed	Section 1.4.1
Prefix A	Appearance checking is allowed	Section 1.4.2
Prefix U	Unordered variant of a grammar class	Section 1.4.2

Lebenslauf des Verfassers

Persönliche Angaben

Name: Frank Weinberg

Staatsangehörigkeit: deutsch

Geburtsdatum: 11.07.1979

Geburtsort: Ludwigshafen

Schul- und Hochschulbildung

25.06.1999	Abitur, Note 1,6
2000 - 2006	Technische Universität Kaiserslautern Diplomstudiengang Informatik
25.07.2002	Voriplom Informatik, Note 1,5
18.05.2006	Diplom Informatik, Note 1,6 Anwendungsfach: Wissensbasierte Systeme / Künstliche Intelligenz

Bisherige Beschäftigungen

01.09.1999 - 30.06.2000	Wehrdienst
01.11.2001 - 28.02.2006	Technische Universität Kaiserslautern, Wissenschaftliche Hilfskraft, Übungsgruppenleitung zu verschiedenen Vorlesungen
16.06.2006 - 15.06.2012	Technische Universität Kaiserslautern, AG Algorithmen und Komplexität, Wissenschaftlicher Mitarbeiter
01.11.2012 - 28.02.2013	Technische Universität Kaiserslautern, Lehrauftrag, Übungen zu Entwurf und Analyse von Algorithmen

Veröffentlichungen

[1] Markus E Nebel, Anika Scheid, and Frank Weinberg. Random generation of RNA secondary structures according to native distributions. *Algorithms for Molecular Biology*, 6(1):1–46, 2011.

[2] Markus E Nebel and Frank Weinberg. Algebraic and combinatorial properties of common RNA pseudoknot classes with applications. *Journal of Computational Biology*, 19(10):1134–1150, 2012.

[3] Frank Weinberg. Position-and-length-dependent context-free grammars. In Jöran Mielke, Ludwig Staiger, and Renate Winter, editors, *Theorietag Automaten und Formale Sprachen*, pages 57–58, 2009.

[4] Frank Weinberg and Markus E Nebel. Extending stochastic context-free grammars for an application in bioinformatics. In *Language and Automata Theory and Applications*, pages 585–595. Springer, 2010.

[5] Frank Weinberg and Markus E Nebel. Non uniform generation of combinatorial objects. Technical report, University of Kaiserslautern, 2010.

[6] Frank Weinberg and Markus E Nebel. Applying length-dependent stochastic context-free grammars to RNA secondary structure prediction. *Algorithms*, 4(4):223–238, 2011.

[7] Frank Weinberg and Georg Zetzsche. Some results on vector grammars with appearance checking. unveröffentlicht.

Die Veröffentlichungen [3], [4] und [6] umfassen jeweils Teile der Ergebnisse dieser Dissertation. Die Ergebnisse aus [7] werden in Kapitel 1.4 referenziert. Die übrigen Arbeiten sind von der Dissertation unabhängige Ergebnisse.