Ambiguity Functions of Context-Free Grammars and Languages

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Klaus Wich

aus Frankfurt/Main

Hauptberichter:	Prof. Dr. V. Diekert
Mitberichter:	Prof. Dr. F. Otto
Mitberichter:	Prof. Dr. J. Esparza
Mitberichter:	Prof. Dr. M. Goldwurm

Tag der mündlichen Prüfung: 3. Dezember 2004

Institut für Formale Methoden der Informatik, Universität Stuttgart

2005

Contents

Chapter 1

Introduction

A context-free grammar G is unambiguous if it does not have two different derivation trees for any word. A context-free language is unambiguous if it is generated by an unambiguous context-free grammar. Context-free grammars and languages are ambiguous if they are not unambiguous. Ambiguous context-free languages are also called inherently ambiguous.[1] The existence of ambiguous context-free languages is shown in [26, 27]. The problem whether or not a context-free grammar or language is ambiguous is undecidable (see [8, 9, 14] or the textbooks [16, Theorem 8.4.5, 8.4.6], [17, Theorem 8.9, 8.16]).

The ambiguity of a word w with respect to a given context-free grammar G is the number of different derivation trees for w generated by G. Ambiguous context-free grammars and languages can be distinguished by their degree of ambiguity, that is, the least upper bound for the ambiguity which a word can have. A context-free grammar is k-ambiguous if k is the least upper bound for the ambiguity of the generated words. A context-free language is k-ambiguous if it is generated by a k-ambiguous context-free grammar but by no $k-1$-ambiguous grammar. For each $k \in \mathbb{N}$ there are examples of k-ambiguous languages [23]. But even languages with an infinite degree of ambiguity exist [30]. A particularly nice example is the Crestin language, which is the square of the (unambiguous) set of palindromes [10], i.e the set of words which can be decomposed as a product of two palindromes.

At first glance this looks like a complete answer to the question which degrees of ambiguity are possible. Since each context-free language is generated by a context-free grammar having finite ambiguity for each particular word, it is quite natural to ask how fast the ambiguity of an infinitely ambiguous

[1] In contrast to many textbooks, we rarely use the word "*inherent*" explicitly since, if applied to a context-free language, the only meaningful interpretation of the word ambiguity is inherent ambiguity.

context-free language grows with respect to the length of the words. The question is probably for the first time addressed in [16, Section 7.1]. Here we can read: "[...] there are inherently ambiguous languages that have an exponential number of derivation trees in the length of the string".[2] The ambiguity function of a context-free grammar maps the natural number n to the maximal ambiguity of words with length at most n. It is non-decreasing. For a context-free grammar we can specify the ambiguity of an arbitrary word. In contrast to that we cannot hold a single word accountable for the ambiguity of a context-free language. In fact, for each context-free language L and each $w \in L$ there is a context-free grammar $G_{L,w}$ generating L and having only one derivation tree for w. One way to prove the ambiguity of a context-free language L nevertheless is to fix an infinite subset L' of L such that each context-free grammar generating L generates all but a finite number of words in L' ambiguously. In such a case the length of the shortest ambiguous word in L' depends on the size of the pumping constant of the considered context-free grammar. Often[3] we can use this dependency on the pumping constant to define an inherent ambiguity function for a context-free language. (See Definition 2.96 for details.)

This thesis examines ambiguity functions for context-free grammars and languages. One major question is which functions are ambiguity functions. Obviously, there are ambiguous context-free grammars for unambiguous context-free languages. In this sense context-free grammars can have superfluous ambiguity. But are there completely superfluous ambiguity functions, i.e., ambiguity functions for context-free grammars which are not inherent for any context-free language? One goal is to characterise the sets of ambiguity functions of context-free languages and grammars as well as possible. Another goal is to examine how different ambiguity classes can be separated. Moreover, we examine how ambiguity is connected to the parsing time of context-free grammars.

It is obvious that useless symbols have no influence on the ambiguity of a context-free grammar. Hence, it is sufficient to consider reduced context-free grammars. Moreover, it is obvious that an ambiguity function of a reduced context-free grammar is ω (infinite) for all but a finite number of arguments if and only if the corresponding grammar is cyclic. A context-free grammar is cyclic if there is a nonterminal which derives itself in a non-void derivation. Since each context-free grammar can be transformed into an equivalent cycle-free context-free grammar, ambiguity functions which reach

[2] A correct example for such a language is presented in [16, Section 7.3]. Unfortunately, the corresponding proof has a gap. This gap has been closed in [24].

[3] It is not clear whether each context-free language has an inherent ambiguity function according to the following definition.

ω cannot be inherent for any context-free languages. Therefore, we only consider cycle-free context-free grammars in the sequel.

A main result of this thesis is that the set of ambiguity functions for cycle-free context-free grammars and the set of ambiguity functions for context-free languages coincide. [4] This reduces the question whether a given function f is an inherent ambiguity function for some context-free language to the corresponding question for cycle-free context-free grammars. Therefore, in order to show that a function f is an inherent ambiguity function for some context-free language L, it is sufficient to find a cycle-free context-free grammar G which is f-ambiguous. In such a proof we do not need to care for the ambiguity of the generated language $L(G)$. This means that, regardless how artificial an ambiguity function $\hat{d}_G$ of a context-free grammar G is for the language $L(G)$, there is always a context-free language L_G which is $\hat{d}_G$-ambiguous.

But how can we characterise the set of ambiguity functions of cycle-free context-free grammars? It is easy to present examples of k-ambiguous and $\Theta(n^k)$-ambiguous context-free grammars for each $k \in \mathbb{N}$. Also exponential ambiguity is easy to achieve. It is clear that super exponential functions are not possible. Moreover, in his diploma thesis (Diplomarbeit) the author already showed that a (cycle-free) context-free grammar is either exponentially ambiguous (i.e., $2^{\Theta(n)}$) or has polynomially bounded ambiguity (i.e., the ambiguity function is within $\mathcal{O}(n^{\Theta(1)})$). This has been achieved by the introduction of an undecidable criterion separating the class of exponentially ambiguous grammars *ECFG* and the grammars with a polynomial bounded ambiguity *PCFG*. It is rather intuitive that a context-free grammar G is in *ECFG* if this criterion is satisfied, but it is quite technical to see that it is in *PCFG* otherwise.

In this thesis we investigate the causes for the gap between *ECFG* and *PCFG* in more detail. This allows us to divide the technical part of the proof into two independent results:

- Firstly, we introduce another intuitive criterion. For context-free grammars which satisfy the new criterion, membership in *PCFG* is easily seen.

- Secondly, we show that the new criterion is complementary to the first one, i.e., each cycle-free context-free grammar satisfies exactly one of the two criteria.

[4] Note that a reduced grammar which has a finite ambiguity for each word is always cycle-free.

Thus, in this thesis the proof of the gap is presented in a more symmetric and accessible way: We have an intuitive sufficient criterion for exponential ambiguity and one for polynomially bounded ambiguity, respectively. Then we show that the two criteria are complementary. As a result both criteria turn out to be also necessary for their respective ambiguity class. The necessity of the new criterion for polynomially bounded ambiguity sheds new light on the relationship between the class of unambiguous context-free languages (*UCFL*) and the class of context-free languages with polynomially bounded ambiguity (*PCFL*), i.e. the class $\{L(G) \mid G \in PCFG\}$. More specifically, *PCFL* turns out to be the closure of *UCFL* under an operation which we call bounded contraction. This result can be applied to generalise a result of Rossmanith and Rytter for parallel parsing from *UCFL* to *PCFL* [28]. Therefore, every language in *PCFL* can be parsed on a CREW-PRAM in logarithmic time.[5] Bounded contraction is a stronger operation on unambiguous languages than union and concatenation together. In fact, the closure of *UCFL* under union and concatenation turns out to be a proper subclass of *PCFL*.

Another interesting result is the discovery of sublinear ambiguity. In fact, we can obtain very slowly growing ambiguity functions. More specifically, for any computable total non-decreasing divergent function f, there is a context-free language L with a divergent ambiguity bounded from above by f. For instance there are inherent divergent ambiguity functions below $\log^*$.

Sublinear ambiguity functions are of interest for sequential parsing. It is well known that the parsing algorithm of Earley [11, 1] parses general context-free grammars in cubic time, while unambiguous context-free grammars are parsed in quadratic time. Earley already improved this result by showing that only a special type of ambiguity, which he calls direct ambiguity, is expensive for his algorithm [11]. He discovered that context-free grammars with a degree of direct ambiguity bounded by a constant have quadratic parsing time, which includes unambiguous context-free grammars and linear context-free grammars.

We introduce an even more specialised form of ambiguity called immediate ambiguity. Immediate ambiguity is bounded by $n + 1$ for words of length n. We show that Earley's algorithm works in time $\mathcal{O}(n^2 \cdot im_G(n))$. Thus, depending on the immediate ambiguity, the costs of parsing range between quadratic and cubic time for each context-free grammar. Moreover, for each reduced context-free grammar G and each $n \in \mathbb{N}$ we have $im_G(n) \leq \hat{d}_G(n + k_G)$, where k_G is a constant only depending on G. Thus,

[5] A CREW-PRAM is a parallel (P) random access machine (RAM). The shared memory is accessed in concurrent read exclusive write (CREW) mode.

for reduced context-free grammars with sublinear ambiguity we obtain subcubic Earley parsing time in general. Immediate ambiguity points out the expensive part of Earley's algorithm more precisely than direct ambiguity. Note that direct ambiguity can be as high as $\Theta(n^{j-1})$ where j is the maximum number of nonterminals on the right-hand side of a production. Despite that the classes of context-free grammars with constantly bounded direct ambiguity and with constantly bounded immediate ambiguity coincide (even though the constant may be lower for immediate ambiguity). Therefore, the difference is not relevant for results on constantly bounded direct ambiguity. This observation also holds for the following result by Earley: Each metalinear grammar is parsed in quadratic parsing time by Earley's algorithm.[6] Let *BDCFL* be the class of languages which can be generated by some context-free grammar with a direct ambiguity bounded by a constant. Note that *BDCFL* contains languages with inherent exponential ambiguity.[7] By a similar argument applied for metalinear languages we will see that the closure of *BDCFL* under union and concatenation too can be parsed in quadratic time by Earley's algorithm. In particular this means that the closure of context-free languages with a finite degree of ambiguity under union and concatenation can be parsed in quadratic time.

As mentioned above we can obtain very slowly growing ambiguity functions far below linear. The corresponding construction can be modified such that the resulting context-free grammars are linear and have an additional property called unambiguous turn position. We can even translate the ambiguity into the number of representatives a regular language R contains for the elements of special trace monoids. This number can be seen as the ambiguity with which the trace is generated by R.

This thesis is structured as follows:

In **Chapter 2** basic notions used in this thesis are introduced. A new formalism is provided which derives trees instead of sentential forms. Since ambiguity is about the number of trees, such a formalism is particularly useful. Trees are denoted here as strings obtained from a preorder traversal (depth first left to right). During this traversal internal nodes are represented by the production applied to them while leaves are denoted by their labels directly. Further, motivation and examples for the new formalisms are provided and some algebraic observations of the tree formalism are proved. For the sake of completeness it is shown that the formalisms are generalisations of the classical ones. Finally, we show that our notions allow a simple alter-

[6] See Definition 2.81 for a definition of metalinear grammars.

[7] Earley shows that the language $\{a^i b^j c^k \mid i = j \text{ or } j = k\}^*$ can be generated by a grammar with bounded direct ambiguity. This language is inherently exponentially ambiguous [24].

native proof for the well-known theorem of Chomsky and Schützenberger on the relation between context-free languages and Dyck languages.

In **Chapter 3** examples for context-free grammars with any finite degree of ambiguity, with polynomial ambiguity of any degree, and with exponential ambiguity are provided. All these ambiguities are obtained by right-linear grammars over a single-letter alphabet.

In **Chapter 4** the costs of some interesting operations with respect to the ambiguity are estimated. This analysis leads to the observation, that many interesting operations have ambiguity costs bounded by polynomials. Thus, the class of languages with polynomially bounded ambiguity *PCFL* is closed under these operations.

Chapter 5 consists of two parts. In the first part we show that an arbitrary reduced context-free grammar G can be parsed in $\mathcal{O}(n^2 \cdot \hat{d}_G(n+k_G))$ time, where k_G is a constant only depending on G but not on n. Provided $\hat{d}_G$ satisfies a very weak homogeneity property, this can be simplified to $\mathcal{O}(n^2 \cdot \hat{d}_G(n))$.[8] Currently an ambiguity function which does not satisfy the required property is not known. In fact, we prove a stronger result which shows that only a special type of ambiguity, which we call immediate ambiguity, is relevant for the parsing time. Immediate ambiguity is a refinement of the notion of direct ambiguity introduced by Earley. But in case of sublinear ambiguity the first estimation is already an improvement compared to the cubic Earley parsing time in the general case. We will see in Chapter 6 that the set of functions addressed with this result is not empty. In fact, ambiguity functions can grow as slowly as any computable function.

The second part of this chapter is dedicated to parallel recognition on a CREW-PRAM. By a straightforward generalisation of a result of Rossmanith and Rytter [28] it can be shown that each language in the closure of *UCFL* under bounded contraction can be recognised on a CREW-PRAM in logarithmic time. In Chapter 7 we will see that this class coincides with *PCFL*.

Chapter 6 is dedicated to sublinear ambiguity. It is shown that for each computable divergent total non-decreasing function $f : \mathbb{N} \to \mathbb{N}$, there is a context-free grammar G with a divergent ambiguity function g, such that $g(n) \leq f(n)$ for all $n \in \mathbb{N}$. This proves that extremely slowly growing divergent inherent ambiguity functions exist. For instance there is a context-free language L with inherently infinite ambiguity below $\log^*$. The construction can be done such that it yields linear context-free grammars which have a so-called unambiguous turn position, i.e., the last step of each

[8] The constant k_G can be omitted if $\exists c \in \mathbb{N} : \forall n \in \mathbb{N} : \hat{d}_G(n+1) \leq c \cdot \hat{d}_G(n)$. An ambiguity function which violates this condition is not known.

derivation is an ε-production and the position where the nonterminal disappears is unique for each generated word. The resulting ambiguity functions can also be transformed into the ambiguity with which a regular language R generates a rational trace language over a special independence alphabet. Here the ambiguity of a trace is the number of its representatives in R.

In **Chapter 7** it is shown that there is no context-free grammar with an ambiguity between exponential and polynomially bounded growth. This result is already part of the diploma thesis (Diplomarbeit) but it is proved here with more specialised formalisms in a less technical and more insightful way. Moreover, with our new proof we see that *PCFL* is the closure of *UCFL* under bounded contractions. Hence, using the results in Chapter 5 we see that each language in *PCFL* can be parsed in logarithmic time on a CREW-PRAM.

Chapter 8 shows that the set of inherent ambiguity functions for context-free languages and the set of ambiguity functions for cycle-free context-free grammars coincide. This result is valuable since we have only a few examples of languages with sublinear ambiguity so far. In contrast to that, Chapter 6 shows that we can create infinitely many sublogarithmic ambiguity functions for context-free *grammars* with growth as slowly as any computable function. With the result of this chapter each of these sublinear ambiguity functions turns out to be inherent for some context-free language. In fact, we prove a somewhat stronger technical theorem, which allows us to use grammars even to show some effects that operations like concatenation or the Kleene star can have on the ambiguity of the resulting languages.

Chapter 9 contains concluding remarks and a list of open problems and conjectures.

Chapter A is a summary in German. (Kapitel **Chapter A** ist eine Zusammenfassung in deutscher Sprache.)

Chapter B contains symbolic notations.

Acknowledgements: I would like to express my deep gratitude to Prof. Dr. Rainer Kemp. His paper [20] was the ignition spark for my Diplomarbeit as well as for my Ph.D. thesis. He also was the advisor of my Diplomarbeit. Even if my visits disturbed him amid his own work he never sent me away. After a few minutes I always had his full attention. His ability to understand abstract ideas at the spot was incredible. He asked many good question and made good suggestions. I deeply regret that he will not receive these thanks anymore, because he passed away on 14.5.2004 much to early aged only 54. The community has not just lost a great scientist but also a very good teacher who cared personally for his students. Beyond that he had a great sense of humour and his most important general advice to handle life was: “Never

take yourself serious", and he never did.

I also would like to thank Prof. Dr. Friedrich Otto who gave me the opportunity to work at his department at the University of Kassel. He and also Gundula Niemann, who was a colleague of mine in Kassel for almost two years, helped me a lot by many valuable discussions. They also improved my English substantially. During my time in Kassel I also had a lot of valuable discussions with Dieter Hofbauer.

The thanks are ordered by my curriculum vitae. Thus, I thank now Prof. Dr. Volker Diekert who was the advisor of my Ph.D. thesis in Stuttgart where I worked for almost 5 years. We had a lot of valuable discussions concerning the ideas of my thesis. Volker Diekert also made a lot of good suggestions concerning the structure of papers and the preparation of talks. This was necessary since five papers made it to international conferences (DLT 99, MFCS 2000, MFCS 2001, ICALP 2002, MFCS 2004) in this fruitful time for me. I also would like to thank my colleagues Holger Austinat, Markus Lohrey, Manfred Kufleitner, Stefan Lewandowsky, Alexander Miller and Holger Petersen for some valuable discussions and Horst Prote for some technical assistance.

I also thank all the referees of my thesis Prof. Dr. Volker Diekert, Prof. Dr. Javier Esparza, Prof. Dr. Friedrich Otto, and Prof. Dr. Massimiliano Goldwurm for reviewing my rather long and in some parts technical thesis. In particular, I want to thank Friedrich Otto and Massimiliano Goldwurm for providing a list of typos and other suggestions which helped to improve the final version of the thesis. I also thank Massimiliano Goldwurm and Alberto Bertoni for some valuable discussions which I had with them on a visit in Milan in 2000. These discussions inspired the consideration of rational trace languages in this thesis.

Finally, I thank my wife Antje. She was extremely patient with the long preparation of this Ph.D. thesis which was planed to be finished before the birth of my daughter Ines. My daughter also tried to help me by being one week late, but she was still by far faster than her daddy. My parents-in-law, Herbert and Brigitte Kramer, also provided a great deal of help by taking care of Ines when this was problematic for Antje and me.

Chapter 2

Preliminaries

This chapter contains most of the definitions used throughout this thesis. It also contains motivation for and basic properties of the newly defined formalisms.

The usual way to define context-free languages by derivations of sentential forms is not suitable for our purpose. The reason is that sentential forms fail to describe derivation trees. If the tree structure is needed, most often the sequence of steps in a leftmost derivation, the so-called left parse, is used. Thus, the loss of information during a derivation is avoided by storing the derivation itself. It is more natural to use a formalism which keeps the relevant information as long as we do not decide to throw it away deliberately. In other words, it is more suitable to derive derivation trees instead of sentential forms. To isolate the frontier we can easily eliminate the internal tree structure whenever we want to. For these reasons a new particularly suitable string representation of trees is introduced in this chapter.

The concepts introduced in this chapter are particularly suitable for ambiguity considerations but they may also be useful beyond the scope of this thesis, for instance in parsing theory.

2.1 Basic Definitions

2.1.1 Sets of Numbers, disjoint union

The set of non-negative integers is denoted by $\mathbb{N}$. Thus, $0 \in \mathbb{N}$. The set of real numbers is denoted by $\mathbb{R}$. The set of positive real numbers is $\mathbb{R}_+ := \{x \in \mathbb{R} \mid x > 0\}$. The *disjoint union* of two sets S_1 and S_2 is denoted by $S_1 \dot{\cup} S_2$.

2.1.2 Quantifiers

If we define a variable i by comparison with a constant without specifying its domain then i is in $\mathbb{N}$. If we write for instance that a statement is true for all $i > 2$ this implicitly means that i is an element of $\mathbb{N}$. Let $S_1, S_2, \ldots S_k$ be arbitrary sets for some $k \in \mathbb{N}$. We write $x_1, \ldots, x_n \in S_1$ for $x_1 \in S_1, \ldots, x_n \in S_1$ where $n \in \mathbb{N}$. We abbreviate $\forall x_1 \in S_1 : \forall x_2 \in S_1 : \cdots : \forall x_n \in S_1 :$ by $\forall x_1, x_2, \ldots, x_n \in S_1$, where $n \in \mathbb{N}$. Moreover, we abbreviate

$$\forall x_{1,1}, \ldots, x_{1,j_1} \in S_1 : \forall x_{2,1}, \ldots, x_{2,j_2} \in S_2 : \cdots : \forall x_{k,1}, \ldots, x_{k,j_k} \in S_k :$$

by

$$\forall x_{1,1}, \ldots, x_{1,j_1} \in S_1,\ x_{2,1}, \ldots, x_{2,j_2} \in S_2,\ \cdots, x_{k,1}, \ldots, x_{k,j_k} \in S_k :$$

where $j_1, \ldots, j_k \in \mathbb{N}$. We use analogous abbreviations for chains of existential quantifiers.

2.1.3 Relations

Let S, T, and U be sets. Let $R_1 \subseteq S \times T$ and $R_2 \subseteq T \times U$ be relations. We often use the infix notation, i.e., we write sR_1t for $(s,t) \in R_1$. We write sR_1tR_2u for sR_1t and tR_2u. The *composition* of R_1 and R_2 is defined by $R_2 \circ R_1 := \{(s,u) \in S \times U \mid \exists t \in T : sR_1tR_2u\}$. The *inverse* of R_1 is $R_1^{-1} := \{(t,s) \in T \times S \mid (s,t) \in R_1\}$. A relation R is said to be *on* S if $R \subseteq S \times S$. Let R be on S. We define the transitive closure R^+ of R by $R^0 := \{(s,s) \mid s \in S\}$, $R^i := R^{i-1} \circ R$ for all $i > 0$. $R^+ := \bigcup_{i=1}^{\infty} R^i$, and the reflexive and transitive closure of R by $R^* := R^+ \cup R^0$. An element $w \in S$ is *irreducible* with respect to R if $(w,u) \notin R$ for all $u \in S$. It is *reducible* otherwise. The relation R is

- *reflexive* if $R^0 \subseteq R$.
- *symmetric* if $R = R^{-1}$.
- *transitive* if $R^+ = R$.
- an *equivalence relation* if R is reflexive, transitive, and symmetric.
- *antisymmetric* if $R \cap R^{-1} \subseteq R^0$.
- a *partial order* if R is reflexive, transitive, and antisymmetric.
- a *linear order* if R is a partial order such that $\forall u, v \in S : uRv$ or vRu holds.

- *noetherian* if there is no infinite sequence $u_0, u_1, \ldots, u_i, \ldots$ of elements in S such that $\forall i \in \mathbb{N} : u_i R u_{i+1}$.
- a *well founded partial order* if R is a partial order and R^{-1} is noetherian.
- *confluent* if $\forall u, v, w \in S : (uR^*v \text{ and } uR^*w \;\Rightarrow\; \exists z \in S : vR^*z \text{ and } wR^*z.)$
- *locally confluent* if $\forall u, v, w \in S : (uRv \text{ and } uRw \;\Rightarrow\; \exists z \in S : vR^*z \text{ and } wR^*z.)$
- *convergent* if it is noetherian and confluent.
- *Church-Rosser*, or said to have the *Church-Rosser property*, if $\forall v, w \in S : (v, w) \in (R \cup R^{-1})^* \;\Rightarrow\; \exists z \in S : vR^*z \text{ and } wR^*z.)$

The latter notions deal with the question under which circumstances it is possible to go in the canonical directed graph defined by R from two words v and w to a common word z along the directed edges. For the Church-Rosser property it suffices when v and w are equivalent with respect to the least equivalence relation containing R. Confluence means that v and w must have a common ancestor along the directed edges. Local confluence means that the common ancestor needed for v and w is a parent of both.

2.1.4 Functions

We extend the linear order $\leq$ on $\mathbb{N}$ to the set $\mathbb{N} \cup \{\omega\}$ by considering ω as the largest element, i.e., we add the pairs in $(\mathbb{N} \cup \{\omega\}) \times \{\omega\}$ to the $\leq$ relation on $\mathbb{N}$. Moreover, $<:=\leq \setminus \{(n, n) \mid n \in \mathbb{N} \cup \{\omega\}\}$.

A function $f : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ is *non-decreasing* if the relation $f(n) \leq f(n+1)$ holds for each $n \in \mathbb{N}$. A non-decreasing function $f : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ is *divergent* if

$$\omega \in f(\mathbb{N}) \text{ or } \forall n \in \mathbb{N} : \exists m \in \mathbb{N} : f(n) < f(m).^1$$

Note that according to this definition the constant function which maps any element of $\mathbb{N}$ to ω is divergent.

Let S and T be sets and $f : S \to T$ be a mapping. The function $f : S \to T$ is *injective* if $f(x) = f(y)$ implies $x = y$ for each $x, y \in S$, it is *surjective* or onto T if for each $t \in T$ there is an $s \in S$ such that $f(s) = t$, and it is *bijective* if it is injective and surjective.

[1] We do not call this property "unbounded" because this definition could lead to strange statements like: "The unbounded function f is bounded by the function g". Note that restricted to non-decreasing functions on $\mathbb{N}$ our notion of divergence agrees with the well known one.

2.1.5 Monoids, Alphabets, and Concatenation

An *operation* $\odot$ on a set M is a mapping $\odot : M \times M \to M$. We write $u \odot v$ instead of $\odot(u, v)$ for each $u, v \in M$. Let M be a set and $\odot$ be an operation on it. The operation $\odot$ is associative if for all $u, v, w \in M$ we have $(u \odot v) \odot w = u \odot (v \odot w)$. A *semigroup* is a pair $(M, \odot)$ consisting of a nonempty set M and an associative operation $\odot$ on M. If a semigroup contains an element $1_M \in M$ satisfying $1_M \odot u = u = u \odot 1_M$ for all $u \in M$ then M is a *monoid* with the *unit* 1_M. If the operation is understood we call M itself a monoid and drop the operation symbol in products. Moreover, if M is understood we write 1 for 1_M. If L is a subset of a monoid M then the closure of L under the operation belonging to M is denoted by L^+. We say that L^+ is the semigroup generated by L. The monoid generated by L is $L^* := L^+ \cup \{1\}$.

A monoid M is *free* over $\Sigma \subseteq M$ if each element of M has a unique representation as a product of elements taken from Σ, i.e., if w is an element of M and if $w = a_1 \cdots a_n = b_1 \cdots b_m$ holds for some $n, m \in \mathbb{N}$, $a_1, \ldots, a_n \in \Sigma$, and $b_1, \ldots, b_m \in \Sigma$, then $n = m$ and $a_i = b_i$ for each $i \in \{1, \ldots, n\}$. For $n = 0$ we have $w = 1$. Note that the case $n \in \{0, 1\}$ is not excluded. For $n = 0$ the element w is the unit of M, and for $n = 1$ the product consists of the single factor a_1. Let M be a free monoid over a nonempty set Σ. Then the set Σ is called the *alphabet* of M, the elements of Σ are called *symbols*, the elements of M are called *words* or *strings*, the unit of M is called *empty word*, and the corresponding operation is called *concatenation*. The empty word is denoted by ε. A word $w \in \Sigma^*$ is called *non-empty* if $w \neq \varepsilon$. Obviously, $M = \Sigma^*$. In the sequel we implicitly define free monoids by specifying their alphabet. Thus, the statement "Let Σ be a finite alphabet" defines the free monoid over Σ. If not stated otherwise we assume alphabets to be nonempty finite sets.

As in many textbooks we implicitly assume symbols to be atomic in the context where they are applied. The next paragraph provides a formalisation of this assumption, which later will be used implicitly:

Whenever we define words over the union of two alphabets Σ_1 and Σ_2 we implicitly assume that Σ_1^* and Σ_2^* are both submonoids of the free monoid over $\Sigma_1 \cup \Sigma_2$. Otherwise it would be possible to represent an element of Σ_1 by words over Σ_2 or vice versa, even if Σ_1 and Σ_2 are disjoint. For instance the free monoid over $\{a, b\}$ contains free submonoids over $\Sigma_1 = \{aa, b\}$ and $\Sigma_2 = \{a, bb\}$, respectively. Thus, Σ_1 and Σ_2 are disjoint sets such that $\Sigma_1 \cap \Sigma_2^+ \neq \emptyset$ and vice versa $\Sigma_2 \cap \Sigma_1^+ \neq \emptyset$. Our convention forbids to call the sets Σ_1 and Σ_2 alphabets in a context where we define words over $\Sigma_1 \cup \Sigma_2$, because $(\Sigma_1 \cup \Sigma_2)^*$ is not free over $\Sigma_1 \cup \Sigma_2$. This convention is applied for

instance when we define the right-hand sides of productions as words over the union of two disjoint alphabets called sets of terminals and nonterminals, respectively. Thus, we implicitly assume that a nonterminal is never a string over the set of terminals and vice versa. Similarly whenever we add a new symbol \$ to an alphabet Σ we assume that Σ^* is a submonoid of the free monoid over $\Sigma \cup \{\$\}$.

2.1.6 Factors of Words, Languages, and Homomorphisms

For arbitrary $i, j \in \mathbb{N}$ the *interval* from i to j is $[i, j] := \{k \in \mathbb{N} \mid i \leq k \leq j\}$. Let Σ be an alphabet. Let $u := a_1 \cdots a_n \in \Sigma^*$ be a word, where $a_i \in \Sigma$ for $1 \leq i \leq n$. The symbol at *position* i is $u[i] := a_i$. The *length* of u is $|u| = n$. The empty word ε is the unique word with length 0. For $i \in [1, n+1]$ and $j \in [0, n]$ the *infix* or *factor* of u from position i to position j is $u[i, j] := a_i \cdots a_j$. If $j < i$ then $u[i, j] = \varepsilon$. We write $u[i, -]$ for $u[i, n]$. A word v is a *prefix* of u if $v = u[1, k]$ for some $k \in [0, n]$. A word v is a *suffix* of u if $v = u[k, -]$ for some $k \in [1, n+1]$. The word u is a *proper prefix* (*proper suffix*) of v if it is a prefix (suffix) such that $u \neq v$.

The *power set* of a set S is $2^S := \{s \mid s \subseteq S\}$. Let Σ be an alphabet. A *formal language* over Σ is a subset of Σ^*. Let L, L_1, and L_2 be formal languages over Σ. Then the *concatenation* of L_1 and L_2 is $L_1 \cdot L_2 := \{uv \in \Sigma^* \mid u \in L_1 \ \wedge \ v \in L_2\}$. We often write $L_1 L_2$ for $L_1 \cdot L_2$. If $w \in \Sigma^*$ then $wL := \{w\}L$ and $Lw := L\{w\}$. The set 2^{Σ^*} is a monoid with the unit $\{\varepsilon\}$. We define $L^0 := \{\varepsilon\}$ and $L^n := L \cdot L^{n-1}$ for $n > 0$. Thus, Σ^n contains all words of length n. The set of words with length up to n is defined $\Sigma^{\leq n} := \cup_{i=0}^{n} \Sigma^i = \{w \in \Sigma^* \mid |w| \leq n\}$. Similarly $\Sigma^{<n} := \cup_{i=0}^{n-1} \Sigma^i$ and $\Sigma^{\geq n} := \Sigma^* \setminus \Sigma^{<n}$. Usually S^n is defined as the n-fold Cartesian product of a set S, i.e., the set of all n tuples over S. But for a language L the expression L^n denotes the set of words which can be written as products of n words in L, which is something different. Whenever we want to form tuples instead of products of words we make this explicit by adding a "$\times$" symbol in the superscript, i.e., $L^{\times n} := \{(w_1, \ldots, w_n) \mid w_1, \ldots, w_n \in L\}$. Note that for a symbol a we have $\{\varepsilon, a\}^2 = \{\varepsilon, a, aa\} \neq \{(\varepsilon, \varepsilon), (\varepsilon, a), (a, \varepsilon), (a, a)\} = \{\varepsilon, a\}^{\times 2}$. In particular $|\{\varepsilon, a\}^2| = 3 \neq 4 = |\{\varepsilon, a\}^{\times 2}|$. A tuple $(w_1, \ldots, w_n) \in (\Sigma^*)^{\times n}$ is a *decomposition* or *factorisation* of the word $w_1 \cdots w_n \in \Sigma^*$.

For $w \in \Sigma^*$ the *reversal* w^R of w is defined by $\varepsilon^R = \varepsilon$ and $(au)^R = u^R a$ for $u \in \Sigma^*$ and $a \in \Sigma$. The reversal is extended wordwise to languages, i.e., for $L \subseteq \Sigma^*$ we define $L^R := \{w^R \mid w \in L\}$.

Let S and T be monoids. A *(monoid) homomorphism* $h : S \to T$ is

a mapping with the property $h(xy) = h(x)h(y)$ and $h(1_S) = 1_T$ for each $x, y \in S$. A homomorphism $h : \Sigma^* \to \Sigma^*$ is uniquely defined by the images of the symbols of Σ. It is *length preserving* if $|h(a)| = 1$ for each $a \in \Sigma$. The *projection* of Σ on a subalphabet Γ is the homomorphism $\pi_{\Sigma\to\Gamma} : \Sigma^* \to \Gamma^*$ given by $\pi_{\Sigma\to\Gamma}(X) := X$ for $X \in \Gamma$ and $\pi_{\Sigma\to\Gamma}(X) = \varepsilon$ for $X \in \Sigma\backslash\Gamma$. Formally $\pi_{\Sigma\to\Gamma}$ depends on Σ but the domain of a projection is always implicitly given by the words on which it is applied. Therefore, we drop the specification of the domain in the sequel thus abbreviating $\pi_{\Sigma\to\Gamma}$ by π_Γ. For each $u \in \Sigma^*, a \in \Sigma$, and $\Gamma \subseteq \Sigma$ we define $|u|_\Gamma := |\pi_\Gamma(u)|$ and $|u|_a := |u|_{\{a\}}$. Let Σ and Γ be two alphabets. We call a homomorphism $\sigma : \Sigma^* \to 2^{\Gamma^*}$ a *substitution*. For $L \subseteq \Sigma^*$ and a substitution σ we define $\sigma(L) := \{u \mid u \in \sigma(w) \text{ for some } w \in L\}$.

Definition 2.1 *For $a, b \in \Sigma$, $L \subseteq \Sigma^*$, and $\tilde{L} \subseteq \Gamma^*$, a* single symbol substitution $[a/\tilde{L}]$ *is a homomorphism $\Sigma^* \to 2^{((\Sigma\cup\Gamma)\backslash\{a\})^*}$ defined by:*

$$[a/\tilde{L}](b) := \begin{cases} \tilde{L} & \text{if } b = a \\ \{b\} & \text{otherwise} \end{cases}$$

We write $L[a/\tilde{L}]$ for $[a/\tilde{L}](L)$.

Example 2.2 *Let $\{a, b, c, \#\}$ be an alphabet. We define the language*

$$L_\diamond := \{w = a^i \# uv \# a^i \mid i \in \mathbb{N}, u, v \in L_p\}$$

where $L_p := \{u \in \{b, c\}^* \mid u = u^R\}$ is the language of palindromes over $\{b, c\}$. Let \$ be a symbol not in $\{a, b, c, \#\}$ and $L := \{a^n \#\$\$\# a^n \mid n \in \mathbb{N}\}$. Then $L_\diamond = L[\$/L_p]$.

2.1.7 Parikh Vectors and Suprema

The *Parikh vector* of a word w over a finite alphabet Σ is a $|\Sigma|$-tuple over $\mathbb{N}$, which has a component for each symbol of Σ. The component associated with an $a \in \Sigma$ is $|w|_a$, which is the number of occurrences of the symbol a in w. Parikh vectors are introduced in [26, 27]. They are frequently used in language theory (for instance see [29].) It is more convenient to use multisets instead of vectors, thus avoiding to specify the correspondence between vector components and letters. We stick to the well known name "Parikh vectors" despite the fact that formally we use multisets.

A *multiset* X over a set S is a mapping $X : S \to \mathbb{N}$. For an $a \in S$ we write $a \in X$ if $X(a) > 0$. We consider $X(a)$ as the number of copies of the element a which are contained in X. A multiset X over S is a *subset* of a multiset Y

over S, denoted by $X \subseteq Y$, if $\forall x \in S : X(x) \leq Y(x)$. The multiset $X \cap Y$ is defined by $(X \cap Y)(a) := \min\{X(a), Y(a)\}$ for each $a \in S$. Similarly $X \cup Y$ is defined by $(X \cup Y)(a) := \max\{X(a), Y(a)\}$ for each $a \in S$. We also compare multisets with sets. A set Z over S is associated with the multiset Z' defined by $Z'(a) = 0$ if $a \notin Z$ and $Z'(a) = 1$ if $a \in Z$. In case of an intersection $Z \cap X$ between a set and a multiset the result can be considered as a set. For a function $f : S \to \mathbb{N}$ we define $\sum_{x \in X} f(x) := \sum_{s \in S} X(s) \cdot f(s)$.

In order to define the supremum of an infinite number of multisets properly we extend $\mathbb{N}$ to a complete lattice by adding a maximal element ω to $\mathbb{N}$. Thus, each set $T \subseteq \mathbb{N}$ has a supremum $\sup(T) \in \mathbb{N} \cup \{\omega\}$.

Definition 2.3 *Let $\mathcal{X}$ be a set of multisets over a finite set S. Then the* supremum *of $\mathcal{X}$ is the mapping $\sup(\mathcal{X}) : S \to \mathbb{N} \cup \{\omega\}$ defined by:*

$$\forall a \in S : \sup(\mathcal{X})(a) \mapsto \sup(\{X(a) \mid X \in \mathcal{X}\}).$$

If $\mathcal{X}$ is finite $\sup(\mathcal{X}) = \cup_{X \in \mathcal{X}} X$. In general suprema of multisets are not necessarily multisets since their range may contain the element ω.

Definition 2.4 *Let w be a word over an alphabet Σ. The* Parikh vector *of w is the multiset $\vec{w} : \Sigma \to \mathbb{N}$ defined by:*

$$\forall a \in \Sigma : \vec{w}(a) \mapsto |w|_a.$$

Definition 2.5 *The* Parikh supremum *of a language L over Σ is defined:*

$$\sup(L) := \sup(\{\vec{w} \mid w \in L\}).$$

The Parikh supremum *of a symbol $a \in \Sigma$ in the language L is $\sup(L)(a)$. For a subalphabet $\Gamma \subseteq \Sigma$ the* Parikh supremum *is defined by the following extension of $\sup(L)$:*

$$\sup(L)(\Gamma) := \sum_{a \in \Gamma} \sup(L)(a).$$

Here the addition on $\mathbb{N} \cup \omega$ is meant in the intuitive way, i.e., $a + b$ is the usual addition on integers if a and b belongs to $\mathbb{N}$ and it is ω otherwise.

An element or subset of Σ is unbounded *in a language L over Σ if its Parikh supremum is ω. It is* bounded *otherwise.*

Intuitively the Parikh supremum of a symbol a in a language L is the maximal number of occurrences of the symbol a in a word of L.

2.1.8 Regular Expressions

Let Σ be an alphabet. Then the syntax of *regular expression* over Σ is recursively defined as follows:

(i) Each $a \in \Sigma$ is a regular expression and ε is a regular expression.

(ii) If α and β are regular expressions then so are $(\alpha \cdot \beta)$, $(\alpha + \beta)$, and (α^*).

For convenience we write $\alpha\beta$ instead of $\alpha \cdot \beta$. We define $\alpha^+ := \alpha\alpha^*$, $\alpha^0 = \varepsilon$, and $\alpha^{n+1} := \alpha\alpha^n$ for each $n \in \mathbb{N}$.

The semantic $L(\alpha)$ of a regular expression α is defined by:

(i) $L(a) = \{a\}$ for each $a \in \Sigma$ and $L(\varepsilon) := \{\varepsilon\}$.

(ii) If α and β are regular expressions then:
$L((\alpha \cdot \beta)) := L(\alpha)L(\beta)$, $L((\alpha + \beta)) := L(\alpha) \cup L(\beta)$, and $L((\alpha^*)) = L(\alpha)^*$.

Parenthesis can be omitted in regular expressions. Then their evaluation is determined by the following precedence rules. As usual paranthesis preceeds over exponents ($*$, $+$, $n \in \mathbb{N}$), exponents precede over products ($\cdot$) and products precede over additions ($+$).

In the sequel we will identify the semantic of regular expressions with their syntax, i.e., for a regular expression α we denote $L(\alpha)$ by α itself. Since we do not deal with the syntax of regular expressions this is no source of confusion.

A language is *regular* if it can be represented by a regular expression.

2.2 Trees and Forests

In this section we introduce tree strings which allow to denote ordered trees and forests in a convenient way. Moreover, we introduce a formalism to derive them in a context-free way. Finally, we show some basic properties of tree strings which represent ordered trees or forests.

2.2.1 Motivation of tree derivations

Readers familiar with context-free languages are likely to expect other formalisms than those found in the subsequent sections. In most textbooks context-free grammars are introduced as tools to generate sentential forms

by a derivation relation. The generated language is defined there as the subset of those sentential forms which consist of terminal symbols only. This approach provides a very simple definition of context-free languages.

The drawback of sentential forms is that they only represent the sequence of leaves but not the tree structure. But ambiguity considerations and parsing theory is about the structure of derivation trees. In case of a non linear context-free grammar even a complete derivation does not always cover the structure of a derivation tree uniquely. For instance if a derivation contains the step:

$$SS \Rightarrow SSS$$

it is not clear which S has been expanded. Parikh solved this problem in his classical report [26] by making the position explicit where the expansion takes place. Another solution is to fix, for each sentential form, the nonterminal which has to be expanded next. For this purpose it is usual to introduce the so-called leftmost derivation, which requires the leftmost nonterminal to be expanded. This leads to a one-to-one correspondence between leftmost derivations and derivation trees. So the ambiguity of a word can be defined as the number of its leftmost derivations. Another choice is to define the ambiguity of a word by the number of its rightmost derivations or any other fixed convention to choose the nonterminal to be expanded next. The resulting number of derivations does not depend on this choice, which is an indication for a sound definition.

But in order to understand the structures which cause ambiguity it is important to consider derivation trees which are not complete, i.e., they may contain nonterminal leaves. Such trees are called embedded trees. A special case of embedded trees are pumping trees, which have a leaf labelled with the same nonterminal as the root. Pumping trees will play a crucial role in Chapter 7.

The most natural way to extend the notion of ambiguity to general sentential forms is to count the number of embedded trees generating them. Unfortunately, some embedded trees fail to have a corresponding leftmost derivation. For instance let us consider a context-free grammar G with the start symbol S and only two productions

$$S \to SS \qquad \text{and} \qquad S \to a,$$

where a is a terminal. The sentential form $SSSa$ has the 5 embedded trees depicted in Figure 2.1. None of them has a leftmost derivation. We say $SSSa$ is not a left sentential form. On the other hand, among them there are exactly three trees, namely T_2, T_4 and T_5, having a rightmost derivation. This example shows that the number of derivations for a given sentential form

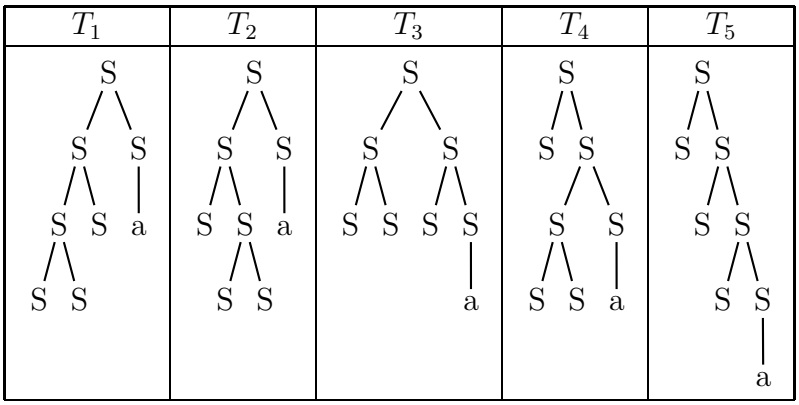


Figure 2.1: Trees for $SSSa$, which only use the productions $[S, SS]$ and $[S, a]$.

crucially depends on the convention for the order in which nonterminals are expanded. But the only meaningful ambiguity of $SSSa$ for the grammar G is not 0, or 3 but 5. So leftmost (and rightmost) derivations are not appropriate for our purposes. We conclude that counting the number of leftmost derivations may be a good way to define the ambiguity for a purely terminal word, but extending this definition to general sentential forms leads to a quite artificial notion of ambiguity.

One way to circumvent this obstacle is to extend the leftmost derivation by marking the nonterminal which has to be expanded next, and add a special rule allowing to skip this nonterminal by moving the marker to the next nonterminal in a left to right order. This has been done in [32]. The main problems can be solved this way but the resulting formalism is still not convenient. For instance a complete leftmost derivation is lengthy and highly redundant since in each step only one symbol is replaced while the remaining part of the sentential form is copied. If we just denote the sequence of applied productions (and skips) in a leftmost order, known as the left parse, it is hard to recover the generated word. The best way to retrieve it, is to draw the derivation tree and read the generated word off the leaves. In other words, the skipping rule introduced in [32] is just another patch for a formalism, which was originally not designed for the investigation of ambiguity.

In some sense Parikh's approach in [26] was more suitable than the modern one. His notion of derivation represents derivation trees for arbitrary sentential forms by making the number of the position explicit where the next symbol is expanded. But in the case of a non-linear context-free grammar (a grammar having more than one nonterminal on the right-hand side of some production) this approach allows several derivations for the same

tree. Parikh dealt with this situation by the use of phrases which belong to a derivation. These phrases are the infixes which correspond to subtrees of the corresponding derivation tree. Since the phrase structure is independent of the order in which neighbouring nonterminals are expanded the problem of multiple derivations for a single tree can be handled this way.

It is even more suitable to derive trees instead of sentential forms from the very beginning, since trees are the central objects to be studied in an investigation on ambiguity. It is important to use a tree representation which allows to extract the corresponding frontier easily. Now sentential forms and left parses are complementary parts of derivation trees. Both can be obtained by a depth first left to right traversal of the tree, also called preorder traversal. On the one hand, for the sentential form we denote the labels at the leaves and ignore the internal nodes. On the other hand, for the left parse of a derivation tree the productions applied to the internal nodes visited are denoted while the leaves are ignored (except they are labelled with a nonterminal, in which case we have to add a skip rule.) It is natural to combine both, to form a single tree formalism, where neither the internal nodes nor the leaves are ignored. The sentential form generated by a tree is easily obtained by cancellation of the internal nodes, and the number of trees generating a sentential form represents its ambiguity no matter whether it consists of terminals only or of a mixture of terminals and nonterminals. The usual derivation relation for context-free grammars can easily be transformed into a tree derivation relation. We just consider the productions themselves as symbols. A production $[A, \alpha]$ is applied by replacing an occurrence of A by a string consisting of the applied production, as the first letter, followed by the right-hand side of the production. By this encoding of trees into words, tree sets become formal languages. The replacement is context-free, and the first symbol of the replaced string characterises the applied production. Hence, the set of derivation trees Δ_G of an arbitrary context-free grammar G is a very simple language, according to the definition in [2]. Very simple languages are a subclass of $LL(1)$ languages which in turn are a subclass of deterministic context-free languages. Thus, we can apply language theoretic results to deal with derivation trees. For instance pumping lemmata for context-free languages apply also to derivation tree sets of context-free grammars. It is even useful to consider the tree generation formalism independently from a given context-free grammar. The task of a context-free grammar G then is to filter out its derivation trees among all the trees generated by the tree expansion relation.

Many important observations on ambiguity can be made independent from a given context-free grammar. In fact, ambiguity can be seen as a property of tree sets. The ambiguity of a context-free grammar G is then just

the ambiguity of G's derivation trees. This approach allows to analyse the ambiguity of subsets of derivation trees. This turns out to be very useful, for example it turns out that a context-free grammar is exponentially ambiguous if and only if its set of pumping trees is ambiguous (see Chapter 8.) It is also used in the proof of Theorem 8.8, which essentially shows that the loss of information induced by an arbitrary length preserving homomorphism can be turned into inherent ambiguity in some sense.

2.2.2 Tree Strings

Trees labelled with elements of a finite alphabet Γ are presented by strings of symbols over the infinite alphabet $\Gamma \cup (\Gamma \times \Gamma^*)$. But not each strings over such an infinite alphabet represent a tree. Some represent a forest while other strings are meaningless. In order to define trees and forests, it is useful to define a derivation relation on arbitrary tree strings first. Intuitively, this derivation allows to transform a leaf into an internal node with leaves attached to it.

Definition 2.6 *Let Γ be a non-empty finite alphabet, called* leaf alphabet. *The infinite alphabet of* internal symbols P_Γ *over Γ is defined by $P_\Gamma := \Gamma \times \Gamma^*$. We denote internal symbols in brackets, i.e., we write $[A, \alpha]$ for the pair $(A, \alpha) \in P_\Gamma$. The infinite* tree alphabet T_Γ *over Γ is defined by $T_\Gamma := \Gamma \cup P_\Gamma$. Elements of T_Γ^* are called* tree strings.

The internal symbols represent internal nodes of trees, where the first component is the label of the node itself and the second is the sequence of labels of its children.

Definition 2.7 *We define the* tree expansion *or* tree derivation $\rightarrow_\Gamma$ *of a non-empty finite alphabet Γ by:*

$$\forall A \in \Gamma, \alpha \in \Gamma^*, \tau_1, \tau_2 \in T_\Gamma^* : \ \tau_1 A \tau_2 \rightarrow_\Gamma \ \tau_1 [A, \alpha] \alpha \tau_2.$$

Let $\tau, \tau' \in T_\Gamma^*$ such that $\tau \overset{*}{\rightarrow}_\Gamma \tau'$, then we call τ' an expansion *of τ and τ a* cut *of τ'.*

Note that in the definition above the pair $[A, \alpha]$ is a single symbol of T_Γ. Thus, $|\tau_1 [A, \alpha] \alpha \tau_2| = |\tau_1 A \tau_2| + |\alpha|$.

Intuitively the relation $\rightarrow_\Gamma$ in the definition above expands a leaf labelled A by turning it into an internal node $[A, \alpha]$ designated to have $|\alpha|$ many children, labelled from left to right with the symbols forming α. The children are actually attached to the internal symbol by writing α to the right of it.

For a unique tree representation it would have been sufficient to denote the arity $|\alpha|$ instead of α itself within internal nodes, but our redundant notation later turns out to be convenient, since it allows to identify the internal node $[A, \alpha]$ with the "production" $[A, \alpha]$ applied to the expanded leaf.

Definition 2.8 *The set of tree strings derived from a tree string $\tau \in T_\Gamma^*$ is defined as:*

$$\Delta_\Gamma^\tau := \{\rho \in T_\Gamma^* \mid \tau \overset{*}{\rightarrow}_\Gamma \rho\}$$

We drop the subscript Γ of $\rightarrow_\Gamma$, P_Γ, T_Γ, and Δ_Γ^τ whenever it is clear from the context.

2.2.3 Roots

In this section it is shown that for each tree string τ there is a unique irreducible tree string $\uparrow(\tau)$ with respect to the tree reduction, which is the inverse of the tree expansion relation. We will call $\uparrow(\tau)$ the *root* of τ. In fact, we show the stronger result that the tree reduction is convergent, i.e., any possible reduction is a first step on a finite path to *the unique* irreducible element.

Definition 2.9 *The* tree reduction *of Γ is the inverse of the tree expansion relation $\rightarrow_\Gamma$, and we denote it by $\leftarrow_\Gamma$, that is, $\leftarrow_\Gamma := (\rightarrow_\Gamma)^{-1}$.*

As for the relation $\rightarrow_\Gamma$ we drop the subscript of $\leftarrow_\Gamma$ if it is clear from the context. For the rest of Section 2.2.3 let Γ be an arbitrary finite alphabet.

Observation 2.10 *Let $\tau_1, \tau_2 \in T^*$ and $i \in \mathbb{N}$. We have*

$$\tau_1 \overset{i}{\rightarrow} \tau_2 \quad \Rightarrow \quad |\tau_1|_P + i = |\tau_2|_P.$$

Proof. The statement can be shown by a trivial induction on i. Obviously, the statement is true for $i = 0$. For $i > 0$ the statement immediately follows from the fact that each step of the tree derivation introduces exactly one internal node. □

Observation 2.10 implies that the tree reduction consumes an internal symbol in each step. This immediately yields:

Observation 2.11 *The tree reduction is noetherian.*

The next lemma shows that the tree reduction is locally confluent. Readers with background in String Rewriting Systems may know that it is sufficient to consider the so called critical pairs of the canonical string rewriting system induced by the relation $\leftarrow_\Gamma$. Since we have not introduced String Rewriting Systems and critical pairs the proof of Lemma 2.12 is written such, that it can be understood without this preknoledge. Implicitly the proof shows that the set of critical pairs is empty: In Case 1 of the proof one sees that no component of a critical pair can be a prefix of the other, since the first letter already specifies the length. In Case 2 one can see that no other overlapping is possible since each component contains exactly one internal symbol, which forms the beginning of the string.

Lemma 2.12 *The tree reduction is locally confluent.*

Proof. Let $u, v, w \in T^*$ such that $u \leftarrow v$ and $u \leftarrow w$. Then there are $l_1, r_1, l_2, r_2 \in T^*$, $A_1, A_2 \in \Gamma$, and $\alpha_1, \alpha_2 \in \Gamma^*$ such that $v = l_1 A_1 r_1$, $w = l_2 A_2 r_2$, and $u = l_1[A_1, \alpha_1]\alpha_1 r_1 = l_2[A_2, \alpha_2]\alpha_2 r_2$.

Case 1 $l_1 = l_2$.

$$u = \begin{array}{|c|c|}\hline l_1 & [A_1,\alpha_1]\alpha_1 r_1 \\ \hline l_2 & [A_2,\alpha_2]\alpha_2 r_2 \\ \hline \end{array} \qquad \Rightarrow \qquad u = \begin{array}{|c|c|c|c|}\hline l_1 & [A_1,\alpha_1] & \alpha_1 & r_1 \\ \hline l_2 & [A_2,\alpha_2] & \alpha_2 & r_2 \\ \hline \end{array}$$

In case $l_1 = l_2$ the symbols $[A_1, \alpha_1]$ and $[A_2, \alpha_2]$ are at the same position and therefore equal. Thus, $A_1 = A_2$ and $\alpha_1 = \alpha_2$ follows. This in turn implies $r_1 = r_2$. Thus, $v = l_1 A_1 r_1 = l_2 A_2 r_2 = w$. Let $z := v$. Then $v \overset{*}{\leftarrow} z$ and $w \overset{*}{\leftarrow} z$ is trivial.

Case 2 $l_1 \neq l_2$. Without loss of generality we assume $|l_1| < |l_2|$. Then $l_1[A_1, \alpha_1]$ is a prefix of l_2. Remember $[A_1, \alpha_1]$ is a single symbol.

$$u = \begin{array}{|c|c|c|}\hline l_1 & [A_1,\alpha_1] & \alpha_1 r_1 \\ \hline \multicolumn{2}{|c|}{l_2} & [A_2,\alpha_2]\ \alpha_2\ r_2 \\ \hline \end{array}$$

Since $[A_2, \alpha_2]$ is an internal symbol and α_1 does not contain internal symbols $l_2 = l_1[A_1, \alpha_1]\alpha_1\tau$ for some $\tau \in T^*$. Thus, $r_1 = \tau[A_2, \alpha_2]\alpha_2 r_2$. Hence, $u = l_1[A_1, \alpha_1]\alpha_1\tau[A_2\alpha_2]\alpha_2 r_2$.

$$u = \begin{array}{|c|c|c|c|c|c|c|}\hline l_1 & [A_1,\alpha_1] & \alpha_1 & \multicolumn{4}{c|}{r_1} \\ \hline l_1 & [A_1,\alpha_1] & \alpha_1 & \tau & [A_2,\alpha_2] & \alpha_2 & r_2 \\ \hline \multicolumn{4}{|c|}{l_2} & [A_2,\alpha_2] & \alpha_2 & r_2 \\ \hline \end{array}$$

Then $v = l_1 A_1 \tau[A_2, \alpha_2]\alpha_2 r_2$ and $w = l_2 A_2 r_2 = l_1[A_1, \alpha_1]\alpha_1\tau A_2 r_2$. With the next reduction step we can get $z := l_1 A_1 \tau A_2 r_2$ from each of the words v and w.

In both cases we found a common descendant (a common ancestor with respect to the expansion relation) of v and w. Hence, $\leftarrow$ is locally confluent. □

In fact we have shown a stronger result: Strings u and v obtained by a local seperation, i.e. u and v are obtained in one reduction step from a common word w, can be merged by the application of a single reduction step to u and v, respectively. This property is known as strong confluence.

Lemma 2.13 *The tree reduction is convergent.*

Proof. The tree reduction is noetherian by Observation 2.11. Therefore, it remains to show that it is also confluent. By Lemma 2.12 the tree reduction is locally confluent. Thus, confluence follows by [7, Theorem 1.1.13]. □

Definition 2.14 *Let ρ be a tree string. The* root *of ρ denoted by $\uparrow(\rho)$ is the unique irreducible tree string τ such that $\rho \in \Delta^{\tau}$.*

Due to Lemma 2.13 roots are well defined. Note that $\tau \in \Delta^{\uparrow(\tau)}$ holds for each tree string $\tau \in T^*$. Moreover, the root mapping is idempotent, i.e., $\uparrow(\uparrow(\tau)) = \uparrow(\tau)$ for each $\tau \in T^*$.

Definition 2.15 *The* single step tree transformation *is the relation:*

$$\leftrightarrow_\Gamma := \rightarrow_\Gamma \cup \leftarrow_\Gamma$$

Again the subscript Γ is dropped whenever it is clear from the context.

Now by [7, Theorem 1.1.7] confluence is equivalent to the Church-Rosser property. Hence, the tree reduction has the Church-Rosser property. According to the definition of the root this implies:

Observation 2.16 *The root of a tree string is preserved by cuts and expansions, i.e., for two tree strings τ_1, τ_2 we have:*

$$(\tau_1 \overset{*}{\leftrightarrow} \tau_2) \Leftrightarrow (\uparrow(\tau_1) = \uparrow(\tau_2)).$$

2.2.4 Trees, Forests and their Visualisation

As already said, tree strings are meant to represent trees and forests. But not all of them have a meaningful interpretation. In this section we introduce the corresponding subsets. Moreover, it is shown how trees can be transformed from a tree string representation into a graphical tree representation and vice versa.

Definition 2.17 *Let Γ be a finite alphabet. Each symbol $X \in T_\Gamma$ of the tree alphabet over Γ has a corresponding* left-hand side $\ell_\Gamma(X)$ *and a corresponding* right-hand side $r_\Gamma(X)$ *defined by:*

$$\ell_\Gamma(X) := \begin{cases} X & \text{if } X \in \Gamma \\ A & \text{if } X = [A, \alpha] \in P \end{cases} \qquad r_\Gamma(X) := \begin{cases} X & \text{if } X \in \Gamma \\ \alpha & \text{if } [A, \alpha] \in P \end{cases}$$

The alphabet Γ forming the subscript of ℓ and r is always identical to the subscript of the tree alphabet T_Γ where the argument X is taken from. Therefore, in the sequel the subscript of ℓ and r is always omitted. The arity *of X is 0 if $X \in \Gamma$ and $|r(X)|$ otherwise.*

Definition 2.18 *Let Γ be a finite alphabet. A string $\rho \in T^*$ is*

- *a* tree *if $\uparrow(\rho) \in \Gamma$.*
- *a* forest *if $\uparrow(\rho) \in \Gamma^*$.*

The set of trees *over Γ is defined by $\Delta_\Gamma := \cup_{A \in \Gamma} \Delta_\Gamma^A$. We drop the subscript Γ when it is clear from the context.*

Let $\rho \in \Delta$ be a tree. The interval $[1, |\rho|]$ is the corresponding set of nodes *and we refer to an i in this interval as the* node *i or the i-th* node *of ρ. The* root node *is the node 1. If i is a node of ρ then $\rho[i]$ is its* parse label*. The node i is a* leaf *if $\rho[i]$ is in Γ, and it is an* internal node *if $\rho[i]$ is in P. The* label *of the i-th node is $\ell(\rho[i])$. The* left-hand side *and the* right-hand side *of the i-th node of ρ are $\ell(\rho[i])$ and $r(\rho[i])$, respectively. The* arity *of the i-th node of ρ is the arity of $\rho[i]$.*

Note that we have not excluded internal nodes with arity 0. This is somewhat unusual, but a natural way to model what is known as ε-productions. So for each $A \in \Gamma$ we can distinguish the trees A and $[A, \varepsilon]$. Both consist of a single node labelled A and having arity 0. But for the tree A this node is a leaf while it is an internal node for the tree $[A, \varepsilon]$. As a consequence the first tree can be expanded by the tree expansion relation, while the second has no more leaf left for an expansion.

Trees can also be represented in a graphical way. Such a representation is easily accessible but it is harder to handle formally. Therefore, we will use the graphical representation only to visualise ideas, while formal proofs are developed by the use of tree strings.

Let $A = \rho_0 \to \rho_1 \to \cdots \to \rho_n$ be a tree expansion with trees $\rho_o, \ldots \rho_n \in \Delta^A$ for some $n \in \mathbb{N}$ and some $A \in \Gamma$. The corresponding trees in their graphical representation are easily obtained as follows: First we draw ρ_0 by writing an A. Now assume we have already drawn ρ_{i-1} for some $i \in$

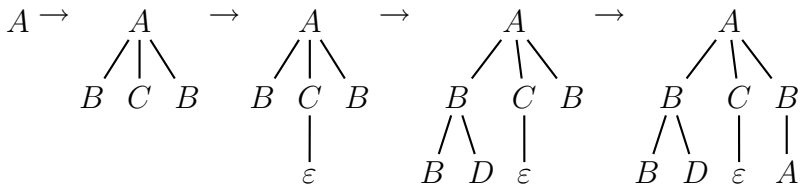


Figure 2.2: Graphical tree derivation.

$[1, n]$. Then there are uniquely defined $\tau_1, \tau_2 \in T^*_\Gamma$, $B \in \Gamma$, and $\alpha \in \Gamma^*$ such that $\rho_{i-1} = \tau_1 B \tau_2$ and $\rho_i = \tau_1[B, \alpha]\alpha\tau_2$. Now let j be by one larger than the number of symbols from Γ in τ_1, i.e., $j := |\pi_\Gamma(\tau_1)| + 1$. The *current leaf* is the j-th proper leaf counted from the left-hand side of the graphical representation of ρ_{i-1}, where a leaf is called *proper* if it is not labelled with ε. We write α beneath the current leaf of ρ_{i-1} and connect each symbol of α to the j-th leaf by a line. In case $\alpha = \varepsilon$ we write ε beneath the current leaf connected by a single line.

Example 2.19 *Let* $\Gamma = \{A, B, C, D\}$. *Then we can derive the tree*

$$\rho := [A, BCB][B, BD]BD[C, \varepsilon][B, A]A \ \in \ \Delta^A.$$

One of the corresponding tree derivations is:

$$\begin{aligned} A &\to [A, BCB]BCB \\ &\to [A, BCB]B[C, \varepsilon]\varepsilon B = [A, BCB]B[C, \varepsilon]B \\ &\to [A, BCB][B, BD]BD[C, \varepsilon]B \\ &\to [A, BCB][B, BD]BD[C, \varepsilon][B, A]A \end{aligned}$$

The graphical representation of this tree derivation is shown in Figure 2.2. The internal nodes of ρ are 1, 2, 5, and 6. The leaves are 3, 4, and 7. Note that node 5 with parse label $[C, \varepsilon]$ is an internal node, with arity 0. The appended ε is formally not a leaf. It is used to distinguish a leaf from an internal node with arity 0.

The tree ρ in the example above could have been derived by several other sequences of trees. If we only have the final tree string ρ we cannot recover the way we obtained ρ but we still have all the information necessary to draw the graphical form of ρ. Assume that ρ is an arbitrary tree string. By the following algorithm we find out whether ρ is a tree, and if it is, we draw the graphical representation of it:

We start by writing down $\ell(\rho[1])$ which we consider as the "current" node. Then we go through ρ symbol by symbol from left to right. If the left-hand side of the considered symbol in ρ does not agree with the label of the current node then ρ is not a tree. Otherwise if the considered symbol in ρ is an internal symbol $[A, \alpha]$ then we write α underneath the current node and draw a line from each symbol of α to the current node. Again if $\alpha = \varepsilon$, we write ε underneath the current node and connect the current node and ε by a single line. In case the considered symbol of ρ is a leaf, we proceed with the next symbol of ρ and shift the current position in the graphical representation to the next proper leaf in a depth first left to right search. In this search we do not consider positions labelled ε as leaves. If all the symbols of ρ are consumed or we cannot find a new current position since we run out of leaves in the graphical representation the algorithm stops. The tree string ρ is a tree if and only if both termination criteria are satisfied simultaneously. If the tree string runs out while we still have a current leaf in the graphical representation then ρ is a prefix of a tree. If we run out of current nodes while ρ is not completely consumed the consumed prefix of ρ was a tree. In this case ρ may be a forest. We can check this by restarting the algorithm with the remaining tree string each time a tree has been successfully consumed.

As already mentioned above we use graphical representations for intuitive explanations while in formal proofs we only use string representations. Thus, there is no need to prove the correctness of the algorithm above. But we proceed with an example of its application:

Example 2.20 *Let* $\Gamma = \{A, B, C, D\}$. *We consider again the tree of Example 2.19:*

$$\rho := [A, BCB][B, BD]BD[C, \varepsilon][B, A]A \ \in \ \Delta^A.$$

We apply the algorithm described above to generate its graphical representation. This process is depicted in Figure 2.3. Beneath a symbol of ρ *one can find the corresponding intermediate tree with the current node underlined.*

Conversely we can obtain the string representation of a tree by going through a graphical representation in a depth first left to right order. For each internal node we write down the corresponding internal symbol while for a leaf we write down the corresponding label.

2.2.5 Properties of Trees and Forests

In order to work with tree strings it is necessary to become familiar with their basic properties and introduce some additional notions as subtrees and embedded trees. Most of the facts proved here are quite obvious but very

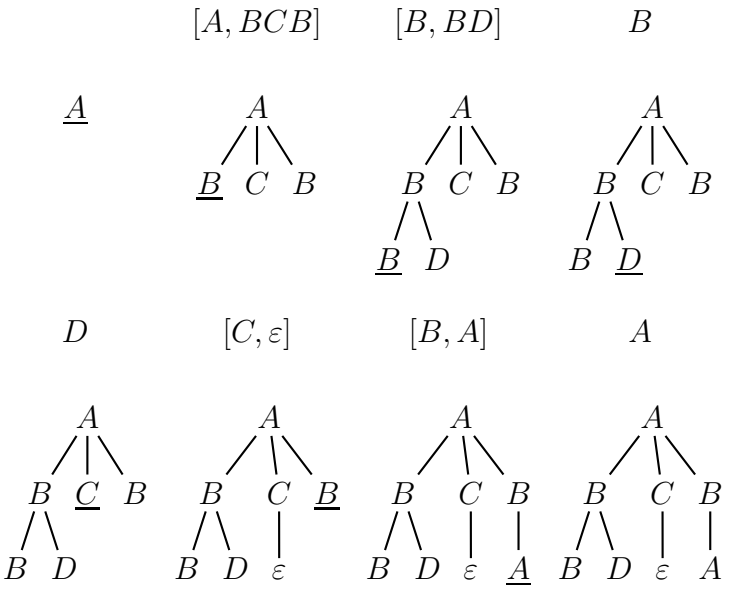


Figure 2.3: Generating the graphical presentation of a tree from a tree string.

important to keep in mind. Statements marked as Observations may be used without reference in later sections. However, within Section 2.2.5 we reference all their applications, to become familiar to them.

Throughout Section 2.2.5 we consider Γ as an arbitrary finite alphabet. Therefore, we drop all subscripts of entities which are parametrised by an alphabet, like P_Γ, T_Γ, and Δ_Γ and write P, T, and Δ instead.

Basic Properties

We begin with some observations covering most elementary properties of roots, trees, and forests. Some of them are self-evident. Despite that at least sketches of their proofs are provided for the sake of completeness. The reader may skip some of them without loss of understanding.

Observation 2.21 *Let $\tau, \tau_1, \tau_2, \omega_1, \omega_2 \in T^*$, $\alpha, \beta \in \Gamma^*$, and $i \in \mathbb{N}$.*

(i) $\omega_1 \xrightarrow{i} \omega_2 \quad \Rightarrow \quad \tau_1\omega_1\tau_2 \xrightarrow{i} \tau_1\omega_2\tau_2$.

(ii) $\uparrow(\tau) = \varepsilon \quad \Leftrightarrow \quad \tau = \varepsilon$.

(iii) $\Delta^\alpha\Delta^\beta = \Delta^{\alpha\beta}$.

(iv) The set of forests over Γ is a monoid with the unit ε generated by the trees over Γ. The corresponding operation is the concatenation on T.

Proof.

(i) The statement can be shown by a trivial induction on i. Obviously, it is true for $i = 0$. For $i > 0$ the statement immediately follows from the fact that, by definition, each step of a tree expansion can also be performed in an extended context.

(ii) Obviously, $\tau = \varepsilon \;\Rightarrow\; \uparrow(\tau) = \varepsilon$. On the other hand $\tau \neq \varepsilon \;\Rightarrow\; \uparrow(\tau) \neq \varepsilon$, since a reduction step never erases a string completely. Hence, the statement follows by a trivial induction on the length of the reduction from τ to $\uparrow(\tau)$.

(iii) Let $\rho \in \Delta^\alpha\Delta^\beta$. Then $\rho = \rho_1\rho_2$ for some $\rho_1 \in \Delta^\alpha$ and $\rho_2 \in \Delta^\beta$. Then by (i) we have $\alpha\beta \xrightarrow{*} \rho_1\beta \xrightarrow{*} \rho_1\rho_2 = \rho$ implying that $\rho \in \Delta^{\alpha\beta}$.

Now let $\rho \in \Delta^{\alpha\beta}$. Then $\alpha\beta \xrightarrow{i} \rho$ for some $i \in \mathbb{N}$. We prove that $\rho \in \Delta^\alpha\Delta^\beta$ by induction on i. For $i = 0$ the statement is trivial. Assume the statement has been proved for some $i \in \mathbb{N}$. Consider the case $\alpha\beta \xrightarrow{i+1} \rho$. By the inductive hypothesis $\alpha\beta \xrightarrow{i} \rho_1\rho_2 \to \rho$ for some $\rho_1 \in \Delta^\alpha$ and $\rho_2 \in \Delta^\beta$. The last step in this derivation expands a *single* symbol *regardless* of the context in which it occurs. This symbol either belongs to ρ_1 or to ρ_2. Without loss of generality we assume that it belongs to ρ_1. Then $\rho_1\rho_2 \to \rho_1'\rho_2 = \rho$ for some $\rho_1' \in T^*$. Obviously, the last step is also possible in the absence of the context ρ_2, i.e., $\rho_1 \to \rho_1'$. Hence, $\rho_1' \in \Delta^\alpha$ implying $\rho \in \Delta^\alpha\Delta^\beta$.

(iv) Let $\rho_1 \in \Delta^\alpha$ and $\rho_2 \in \Delta^\beta$ then $\rho_1\rho_2 \in \Delta^\alpha\Delta^\beta \overset{\text{(iii)}}{=} \Delta^{\alpha\beta}$. Hence, the concatenation of two forests is a forest. Obviously, ε is the unit and $\uparrow(\varepsilon) \overset{\text{(ii)}}{=} \varepsilon \in \Gamma^*$. Hence, ε is a forest. It remains to show that each forest is represented by a finite product of trees. Let ρ be a forest over Γ. Then for some $k \in \mathbb{N}$ and $X_1, \ldots X_k \in \Gamma$ we have $\rho \in \Delta^{X_1 \cdots X_k}$. By iterated applications of (iii) we verify that $\Delta^{X_1 \cdots X_k} = \Delta^{X_1} \cdots \Delta^{X_k}$. Hence, ρ is a sequence of trees.

□

Note that Observation 2.21 (iv) justifies to write the set of all forests by Δ^*. We will see later that Δ^* is the *free* monoid over Δ.

Observation 2.22 *Let $\tau, \tau_1, \tau_2, \tau_3 \in T^*$, and $\alpha \in \Gamma^*$. Then we have:*

(i) $\uparrow(\tau_1\tau_2\tau_3) = \uparrow(\tau_1\uparrow(\tau_2)\tau_3)$.

(ii) $\uparrow(\alpha) = \alpha$.

(iii) $\uparrow(\alpha\tau) = \alpha\uparrow(\tau)$.

Proof. (i) Immediate consequence of Observation 2.16.

(ii) With $|\alpha|_P = 0$ and Observation 2.10 we obtain that α is irreducible, i.e., $\uparrow(\alpha) = \alpha$.

(iii) Assume $\alpha\uparrow(\tau)$ is reducible. Then $\alpha\uparrow(\tau) = \omega_1[B,\beta]\beta\omega_2$ for some $\omega_1, \omega_2 \in T^*$, $B \in \Gamma$, and $\beta \in \Gamma^*$. Now the strings α and $\uparrow(\tau)$ are irreducible. This is due to (ii) and the definition of the root, respectively. Hence, $[B,\beta]\beta$, which can be reduced to B is neither an infix of α nor an infix of $\uparrow(\tau)$, i.e., ω_1 is a proper prefix of α and ω_2 is a proper suffix of $\uparrow(\tau)$.

This situation can be illustrated as follows:

α		$\uparrow(\tau)$	
ω_1	$[B,\beta]$	β	ω_2

Thus, under our assumption $[B,\beta]$ lies within α, which is impossible, since $\alpha \in \Gamma^*$ and $[B,\beta] \notin \Gamma$. Therefore, the assumption is false and we obtain that $\alpha\uparrow(\tau)$ is irreducible, i.e., $\alpha\uparrow(\tau) = \uparrow(\alpha\uparrow(\tau))$. Finally, we observe $\uparrow(\alpha\tau) \overset{\text{(i)}}{=} \uparrow(\alpha\uparrow(\tau)) = \alpha\uparrow(\tau)$.

□

Bottom Up Tree Description and Noetherian Induction

The tree expansion replaces a leaf by an internal node with new appended leaves. This "context-free" top down rewriting is very similar to the usual derivation relation for context-free grammars. There is however an equivalent way to consider trees in a bottom up manner, i.e., one can add a single production atop a forest to create a tree. Sometimes this representation allows convenient proofs by noetherian induction.

Definition 2.23 *Let ρ and ρ' be trees over Γ such that $\rho = \tau_1\rho'\tau_2$ for some $\tau_1, \tau_2 \in T^*$. Then ρ' is a* subtree *of ρ. It is a* proper subtree *of ρ if $\tau_1\tau_2 \neq \varepsilon$.*

Obviously, the subtree relation is well founded. Hence, it admits proofs by noetherian induction:

Observation 2.24 *To prove that a predicate R holds for each tree $\rho \in \Delta$ it is sufficient to show that the truth of R for each proper subtree of ρ implies the truth of R for ρ.*

In order to use noetherian induction we need to know more about the distribution of subtrees within trees.

Observation 2.25 *Let $\rho \in \Delta$ be a tree. Then*

(i) ρ is irreducible if and only if $\rho \in \Gamma$.

(ii) ρ is reducible if and only if $\rho = [\uparrow(\rho), \alpha]\tau$ for some $\alpha \in \Gamma^*$ such that $\tau \in \Delta^\alpha$. [2]

(iii) ρ is reducible if and only if $\rho = [\uparrow(\rho), X_1 \cdots X_k]\rho_1 \cdots \rho_k$ for some $k \in \mathbb{N}$, $X_1, \ldots, X_k \in \Gamma$, and $\rho_i \in \Delta^{X_i}$, for each $i \in [1, k]$.

Proof. (i) By Definition 2.18 we have $\uparrow(\rho) \in \Gamma$. Hence, ρ can only be irreducible if $\rho \in \Gamma$. On the other hand if $\rho \in \Gamma$ then ρ is irreducible by Observation 2.22(ii).

(ii) By Definition 2.18 we have $\uparrow(\rho) \in \Gamma$. Thus, ρ is obviously reducible if ρ is of the form $[\uparrow(\rho), \alpha]\tau$, which implies $\uparrow(\rho) \to \omega \xrightarrow{*} \rho$, for some $\omega \in T^*$. Now $\uparrow(\rho) \in \Gamma$ implies $\omega = [\uparrow(\rho), \alpha]\alpha$ for some $\alpha \in \Gamma^*$. Moreover, $[\uparrow(\rho), \alpha]$ cannot be expanded and remains the first symbol during a derivation of ρ from ω. Hence, a trivial induction on the length of the derivation yields $\rho = [\uparrow(\rho), \alpha]\tau$ for some $\tau \in \Delta^\alpha$.

(iii) Immediate consequence of Observations 2.21(iv) and 2.25(ii).

□

Sometimes it is not necessary to be as specific as in Observation 2.25. Since ε is a forest (Observation 2.21(iv)), an inspection of Observation 2.25 yields a weaker but less technical fact:

Corollary 2.26 *Each tree consists of a single symbol followed by a sequence of proper subtrees, i.e., $\Delta \subseteq T\Delta^*$.*

Note that $\Delta \neq T\Delta^*$, e.g., $[A, B]A \in T\Delta^* \setminus \Delta$ for $A, B \in \Gamma$.

Forests are Free over Trees

Here we show that no nonempty suffix of a tree can ever be a proper prefix of a tree. This implies that forests are free monoids over trees. Therefore, the decomposition of reducible trees according to Observation 2.25(iii) is unique. We already know that a forest can only be generated by a sequence of trees whose roots form the root of the forest.

[2] Note that this implies that the root is the label of the root node.

Lemma 2.27 *Each suffix of a tree is a forest.*

Proof. We prove the statement by noetherian induction. We assume that $\rho \in \Delta$ is an arbitrary tree such that each suffix of each proper subtree is a forest. Let $\tau \in T^*$ be a suffix of ρ. If $\tau = \rho$ then τ is a tree and therefore also a forest according to Definition 2.18. It remains to prove the statement for proper suffices τ of ρ. According to Corollary 2.26 and Observation 2.21(iv) we get $\rho = X\rho_1 \cdots \rho_k$ for some $X \in T$, and some $\rho_1, \ldots, \rho_k \in \Delta$ which are proper subtrees of ρ. Since τ is a proper suffix of ρ, we have $\tau = \tau'\rho_{i+1} \cdots \rho_k$ for some $i \in [1, k]$ and some suffix τ' of ρ_i. Then τ' is a forest by the inductive hypothesis. On the other hand $\rho_{i+1} \cdots \rho_k$ is a forest too. Thus, by Observation 2.21(iii) the string τ is a forest. □

Lemma 2.27 turns out to be essential for the understanding of forest factorisations. Now it is a matter of taste whether one prefers an algorithmic, or a more succinct but less constructive algebraic approach to finish the proof that Δ^* is free over Δ. We proceed in the later way. In an appended paragraph on page 38 the reader who prefers the first way finds a good algorithm to factorise forests. Such a reader is recommended to continue there, and return later to read Lemma 2.28, Corollary 2.29, and Corollary 2.30, which will be evident by then without the algebraic proof.

Lemma 2.28 *No proper prefix of a tree is a nonempty suffix of a tree, i.e., if $\tau_1\omega \in \Delta$ and $\omega\tau_2 \in \Delta$ for some $\tau_1, \omega, \tau_2 \in T^*$ then either $\omega = \varepsilon$ or $\tau_2 = \varepsilon$.*

Proof. Let $\tau_1, \omega, \tau_2 \in T^*$ such that $\tau_1\omega \in \Delta$ and $\omega\tau_2 \in \Delta$. Since ω is a suffix of the tree $\tau_1\omega$ and τ_2 is a suffix of the tree $\omega\tau_2$, Lemma 2.27 implies that ω and τ_2 are forests. Moreover, $\uparrow(\omega\tau_2) = A$ for some $A \in \Gamma$ since $\omega\tau_2 \in \Delta$ is a tree. By definition $\omega \in \Delta^{\uparrow(\omega)}$ and $\tau_2 \in \Delta^{\uparrow(\tau_2)}$. Hence, by Observation 2.21(iii) we have $\omega\tau_2 \in \Delta^{\uparrow(\omega)}\Delta^{\uparrow(\tau_2)} = \Delta^{\uparrow(\omega)\uparrow(\tau_2)}$, i.e., $A = \uparrow(\omega\tau_2) = \uparrow(\omega)\uparrow(\tau_2)$. Hence, $\uparrow(\omega) = \varepsilon$ or $\uparrow(\tau_2) = \varepsilon$ and by Observation 2.21(ii) we finally get $\omega = \varepsilon$ or $\tau_2 = \varepsilon$. □

Corollary 2.29 *No proper prefix of a tree is a tree.*

Proof. Let $\omega \in \Delta$ be a tree which is a (not necessarily proper) prefix of a tree $\omega\tau_2 \in \Delta$ for some $\tau_2 \in T^*$. Setting $\tau_1 := \varepsilon$ we have $\tau_1\omega \in \Delta$ and $\omega\tau_2 \in \Delta$. Then Lemma 2.28 implies $\omega = \varepsilon$ or $\tau_2 = \varepsilon$. Since $\omega \in \Delta$ we have $\omega \neq \varepsilon$ (Corollary 2.26). Hence, $\tau_2 = \varepsilon$ and therefore ω is not a proper prefix of $\omega\tau_2$. □

Since forests are generated by trees (Observation 2.21(iv)) and trees are a prefix code for forests according to Corollary 2.29 we get:

Corollary 2.30 *The set of forests Δ^* is the free monoid over the set of trees Δ with the unit ε.*

Factorising Forests

In Section 2.2.4 we already suggested an algorithm (without a proof) to factorise a forest by drawing the corresponding graphical representation. This method was also capable to detect whether or not a tree string is a valid forest. If we already know that a tree string is a forest there is however a much simpler way to split the forest into the corresponding sequence of trees. Due to their close relation to balanced parenthesis we can detect the first tree of a forest as the first prefix having weight 1 according to an appropriate weight function w for the symbols in T.

Definition 2.31 *The function $w : T^* \to \mathbb{Z}$ is a homomorphism where the operation on $\mathbb{Z}$ is the usual addition on integers. That is, the function w sums up weights of symbols in T, which are defined as follows:*

$$w(X) := \begin{cases} 1 & \text{if } X \in \Gamma \\ 1 - |r(X)| & \text{if } X \in P. \end{cases}$$

Remember $r(X)$ is the right-hand side of the internal symbol X. For each $\chi \in T^*$ we call $w(\chi)$ the weight *of χ.*

Obviously, irreducible trees have weight 1. The proof of the next Lemma shows that the weight is preserved by tree expansions.

Lemma 2.32 *Each tree has weight 1, i.e., $w(\rho) = 1$ for each $\rho \in \Delta$.*

Proof. Let $\rho \in \Delta$. Then for some $A \in \Gamma$ and some $i \in \mathbb{N}$ we have $A \xrightarrow{i} \rho$. We prove $w(\rho) = 1$ by induction on i. For $i = 0$ the statement is trivial. Assume the statement is true for each tree with a derivation of length i. Assume $A \xrightarrow{i+1} \rho$. Then for some $[B, \beta] \in P$ and some $\tau_1, \tau_2 \in T^*$ we have $A \xrightarrow{i} \tau_1 B \tau_2 \to \tau_1[B, \beta]\beta\tau_2 = \rho$. By the inductive hypothesis $w(\tau_1 B \tau_2) = 1$. Thus, $w(\rho) = w(\tau_1)+w([B, \beta])+w(\beta)+w(\tau_2) = w(\tau_1)+(1-|\beta|)+|\beta|+w(\tau_2) = w(\tau_1) + 1 + w(\tau_2) = w(\tau_1) + w(B) + w(\tau_2) = w(\tau_1 B \tau_2) = 1$. □

Thus, intuitively, the mapping w applied to a forest τ "counts" the number of trees "in" τ, i.e. $w(\tau) = |\uparrow(\tau)|$.

Lemma 2.33 *Any proper prefix τ of a tree $\rho \in \Delta$ has a weight smaller than 1, i.e., $w(\tau) < 1$*

Proof. If τ is a proper prefix of ρ then $\rho = \tau\tau'$ for some $\tau' \in T^+$. But then τ' is a non-empty product of trees by Lemma 2.27 and Observation 2.21(ii). Thus, $w(\tau') \geq (1)$. Therefore, $w(\tau) = w(\tau) + w(\tau') - w(\tau') = w(\tau\tau') - w(\tau') = w(\rho) - w(\tau') \overset{2.32}{=} 1 - w(\tau') < 1$. □

Lemma 2.32 and Lemma 2.33 imply that the shortest prefix with weight 1 marks the end of the first tree in a forest. Therefore, to factorise a forest into its sequence of trees it suffices to compute the weights of all prefixes incrementally from left to right.

We can consider Lemma 2.32 and Lemma 2.33 as an alternative proof for Corollary 2.29 and Corollary 2.30. Moreover, as we have seen in the proof of Lemma 2.33 each non-empty suffix of a tree has a weight of at least 1 and can therefore not be a proper prefix of a tree (Lemma 2.33). Thus, we can also prove Lemma 2.28 by considering weights of strings.

Substructures of Trees

Definition 2.34 *Let $\rho \in T^*_\Gamma$. The* frontier *or* yield mapping *is the projection $\downarrow_\Gamma := \pi_\Gamma$, i.e., the frontier or yield of ρ is $\downarrow_\Gamma(\rho)$.*

In the sequel we drop the subscript of $\downarrow_\Gamma$ because Γ is always implicitly given by the domain of the string on which the mapping is applied. For the tree of Example 2.20 we obtain:

$$\downarrow(\rho) = \downarrow\big([A, BCB][B, BD]BD[C, \varepsilon][B, A]A\big) = BDA.$$

The frontier is the result of reading the leaves from left to right. In the graphical representation of a tree the frontier is located at the bottom. That's the reason why the symbol for the frontier mapping "$\downarrow$" points downward. For the same reason the symbol for the root mapping "$\uparrow$" points upward.

Definition 2.35 *A* phrase *of $\rho \in \Delta$ is an interval $[i, j]$ such that $\rho[i, j]$ is a subtree of ρ. This subtree is called the* tree belonging to phrase $[i, j]$. *Then the word $\rho[1, i-1] \cdot \uparrow(\rho[i, j]) \cdot \rho[j+1, |\rho|] \in \Delta$ is called the* remainder tree *obtained by* truncation *of the phrase $[i, j]$.*

Whenever we apply tree notions to a phrase $[i, j]$ of a tree ρ we mean to apply them to the tree $\rho[i, j]$ belonging to $[i, j]$. For instance we may talk about the root or frontier of the phrase $[i, j]$ of ρ and mean $\uparrow(\rho[i, j])$ and $\downarrow(\rho[i, j])$, respectively.

Observation 2.36 *Let $\rho \in \Delta$ and $i \in [1, |\rho|]$. Then there is a unique $j \in [i, |\rho|]$ such that $[i, j]$ is a phrase.*

Proof. Since each suffix of a derivation tree is a forest (Lemma 2.27), each non empty suffix of a tree consists of at least one tree (Observation 2.21(ii) and Observation 2.21(iv)). Since trees are a prefix code for forests (Corollary 2.29) the end of the first tree is uniquely defined. □

As an immediate consequence of observation 2.36 we obtain:

Corollary 2.37 *Each derivation tree $\rho \in \Delta$ has exactly $|\rho|$ phrases.*

Observation 2.36 implies that the following definition is well defined.

Definition 2.38 *The i-th* phrase *of a derivation tree ρ for an $i \in [1, |\rho|]$ is $[i, j]$ for the uniquely defined $j \in \mathbb{N}$ such that $[i, j]$ is a phrase of ρ. The tree $\rho[i, j]$ is called the* tree attached to node i.

Note that the number of subtrees of a tree may be smaller than the number of phrases. This happens when the same subtree is attached to different nodes.

Definition 2.39 *Let $\rho \in \Delta$ and $i, j \in [1, |\rho|]$. The node i is an* ancestor *of j and j is a* descendant *of i if the j-th phrase is a subset of the i-th phrase. In case i is an ancestor (descendant) of j, we call i a* proper ancestor *(a* proper descendant*) if $i \neq j$. We say that i and j are* related *if i is an ancestor or descendant of j. They are called* independent *if the i-th and j-th phrase are disjoint.*

As an immediate consequence of Lemma 2.28 we obtain:

Observation 2.40 *If i and j are two nodes of a tree then they are either related or independent.*

Definition 2.41 *Let $\rho \in \Delta$ and $i, j \in [1, |\rho|]$. The node i is called* parent *of j and j is called* child *of i if i is the shortest proper ancestor of j.*

Observation 2.42 *Each node of a tree has at most one parent.*

Proof. Assume to the contrary that a node i of a tree has two different parents j and k. Then the j-th and the k-th phrases both contain i hence j and k are not independent. On the other hand both phrases have the same number of elements and they are different. Thus, they are not comparable with respect to the subset relation. Hence, they are not related either, which contradicts Observation 2.40. □

Note that the node 1 does not have a parent. However, $[1, |\rho|]$ is obviously a phrase if ρ is a tree. Therefore, each node except 1 has a proper ancestor. Similarly each node which is not a leaf has at least one child.

The path to a node i in a tree ρ is obtained by replacing the subtree attached to i by ε and reducing all phrases which do not overlap with i.

Definition 2.43 *Let $\rho \in \Delta$ be a tree. The* path *to the node i is:*

$$\uparrow(\tau_1)\,[\uparrow(\rho[i,j]), \varepsilon]\,\uparrow(\tau_2)$$

where $\rho = \tau_1\rho[i,j]\tau_2$ and $\rho[i,j] \in \Delta$ is the tree attached to node i.

A path is a tree such that each node has at most one child which is not a leaf. A straight forward proof shows:

Observation 2.44 *The mapping of a node to its path is injective.*

Definition 2.45 *Let i be a node of a tree ρ and j a node of a tree ρ'. Then we say that the i-th node of ρ is equivalent to the j-th node of ρ' if the path to i in ρ and the path to j in ρ' coincide.*

Let $\mathcal{P}$ be a property of nodes. The node μ is the *first ancestor* of ν with property $\mathcal{P}$ if no proper descendant of μ which satisfies $\mathcal{P}$ is an ancestor of ν. The *first common ancestor* of two nodes ν_1 and ν_2 is the first ancestor of ν_1 which is an ancestor of ν_2. A node ν is a child of a node μ if μ is the first proper ancestor of ν.

2.2.6 Tree Languages

We already defined trees and forests as subsets of strings over a tree alphabet T_Γ which in turn is defined by an alphabet Γ. In this sense trees and forests are languages over the alphabet T_Γ, which is not finite if $\Gamma \neq \emptyset$. In this section we extend the already defined notions of cuts and subtrees to tree languages and introduce the notion of *embedded* trees.

Definition 2.46 *A* tree language *L over Γ is a subset of Δ_Γ.*

Definition 2.47 *Let $L \subseteq \Delta$. Then*

$$\begin{aligned} \mathit{subtree}(L) &:= \{\rho \in \Delta \mid \exists \rho' \in L : \rho \text{ is a subtree of } \rho'\}. \\ \mathit{cut}(L) &:= \{\rho \in \Delta \mid \exists \rho' \in L : \rho \text{ is a cut of } \rho'\}. \\ \mathit{embedded}(L) &:= \mathit{cut}(\mathit{subtree}(L)). \end{aligned}$$

A single tree $\rho \in \Delta$ is treated as the singleton set $\{\rho\}$, e.g., we write $\mathit{cut}(\rho)$ for $\mathit{cut}(\{\rho\})$. A tree $\rho' \in \Delta$ is embedded *in a tree $\rho \in \Delta$ if $\rho' \in \mathit{embedded}(\rho)$.*

Note that $\mathit{embedded}(L)$ is the least set of trees $L' \subseteq \Delta$ which contains L and is closed under cuts and subtrees, i.e., it is the least set with the property:

$$L \subseteq L' \quad \text{and} \quad \mathit{cut}(L') = L' \quad \text{and} \quad \mathit{subtree}(L') = L'.$$

Example 2.48 *Let* $\Gamma = \{A, B, C, D, S\}$. *The tree*

$$\rho = [S, AB][A, CAC]C[A, D]DCB$$

has the following set of subtrees:

$$\{B,\ C,\ D,\ [A, D]D,\ [A, CAC]C[A, D]DC,\ \rho\}.$$

According to Definition 2.7 the set of ρ*'s cuts is:*

$$\{\rho,\ [S, AB][A, CAC]CACB,\ [S, AB]AB,\ S\}$$

All cuts and subtrees of ρ *are trees embedded in* ρ. *In addition,* ρ *contains the two embedded trees* A *and* $[A, CAC]CAC$, *both of which are cuts of the subtree* $[A, CAC]C[A, D]DC$.

2.3 Context-Free Grammars and Languages

2.3.1 Basic Notations

Now that we have a tree generation formalism we introduce context-free grammars as tools to filter out the relevant trees. The generated language is then filtered out of the raw material of trees by the frontier mapping $\downarrow$.

Definition 2.49 *A* context free grammar *is a quadruple* $G = (N, \Sigma, P, S)$, *where the sets* N *and* Σ *are two disjoint finite alphabets called* nonterminals *and* terminals, *respectively,* $P \subseteq N \times (N \cup \Sigma)^*$ *is a finite alphabet called* productions, rules, *or* internal symbols *and* $S \in N$ *is the* start symbol. *The* tree alphabet *of* G *is* $T_G := N \cup \Sigma \cup P$. *Note that* T_G *is a finite subset of the infinite alphabet* $T_{N\cup\Sigma}$. *Let* $A \in N$. *We define:*

- $\Delta_G := \Delta^S_{N\cup\Sigma} \cap (P \cup \Sigma)^*$ *the set of* G*'s* derivation trees.
- $L(G) := \downarrow(\Delta_G)$ *the* language generated by G. *A word* $w \in \Sigma^*$ *is said to be* generated by G *if* $w \in L(G)$.
- *cut*(Δ_G) *the set of* G*'s* sentential derivation trees.
- $\mathcal{S}_G := \downarrow(cut(\Delta_G))$ *the set of* sentential forms *generated by* G.
- $\Lambda_G := \{\rho \in embedded(\Delta_G) \mid \downarrow(\rho) \in \Sigma^*\uparrow(\rho)\Sigma^*\}$ *the set of* G*'s* pumping trees. *The* link node *of a pumping tree* $\lambda \in \Lambda_G$ *is the root node if* λ *consists of one node only, otherwise it is the unique leaf which is labelled by a nonterminal. (Note that* $\Sigma \subseteq \Lambda_G$.*)*

Definition 2.50 *A language $L \subseteq \Sigma^*$ over some finite alphabet Σ is* context-free *if $L = L(G)$ for some context-free grammar G.*

Two context-free grammars G_1 and G_2 are called *equivalent* if $L(G_1) = L(G_2)$.

The set of derivation trees of G is the set of trees which are an expansion of the start symbol where only productions in P have been applied and each leaf is labelled with a terminal. The language generated by G is the set of frontiers of the derivation trees. The sets of sentential derivation trees and sentential forms are defined analogously to derivation trees and the generated language but the former may still contain nonterminals.[3]

Due to the topic of this thesis we are rarely interested in sentential forms without any access to the tree which generates them. But in these rare cases it is convenient to use the classical derivation relation on sentential forms:

Definition 2.51 *Let $G = (N, \Sigma, P, S)$ be a context-free grammar. We define the* derivation relation $\Rightarrow_G^*$ *of G as the reflexive and transitive closure of the single step derivation relation $\Rightarrow_G$ which is defined by:*

$$\forall \alpha, \gamma \in (N \cup \Sigma)^*, [B, \beta] \in P : \ \alpha B \gamma \Rightarrow_G \alpha \beta \gamma.$$

2.3.2 Short Notation

In the sequel we often abbreviate the definition of a context-free grammar by specifying its production set only. In these cases the nonterminal and terminal sets will consist of Roman letters and Arabic digits only. The set of nonterminals is implicitly given by the symbols which occur on the left-hand side of some production. The remaining symbols occurring on the right-hand sides of productions are the terminals. The start symbol is the left-hand side of the first production occurring in the notation of the production set. We abbreviate $A \to \alpha_1, \ldots, A \to \alpha_k$ by $A \to \alpha_1 | \cdots | \alpha_k$. Lists of productions for different nonterminals are separated by ", " or a new line.

Example 2.52 *The context-free grammar:*

$$G := (\{S, A\}, \{a, b\}, \{(S, AA), (A, aAa), (A, bAb), (A, a), (A, b), (A, \varepsilon)\}, S)$$

can be written as:

$$G := \ (S \to AA, \quad A \to aAa \mid bAb \mid a \mid b \mid \varepsilon)$$

[3] Terminal or nonterminals can only occur as leaves in a tree, since the parse label of an internal node is always a production.

2.3.3 Production Types

Let $G = (N, \Sigma, P, S)$ be a context-free grammar. A production $[A, \alpha] \in P$ can also be denoted as $A \to \alpha$ or $[A \to \alpha]$. It is called a

- *chain production* if $\alpha \in N$.
- *terminal production* if $\alpha \in \Sigma^*$.
- *ε-production* if $\alpha = \varepsilon$.
- *Greibach production* if $\alpha \in \Sigma N^*$.
- *linear production* if $\alpha \in \Sigma^*(N \cup \{\varepsilon\})\Sigma^*$
- *right linear production* if $\alpha \in \Sigma^*(N \cup \{\varepsilon\})$.

The grammar G is *ε-free* if no tree in $\Delta_G \setminus \{[S, \varepsilon]\}$ contains ε-productions, it is in *Greibach normal form*, *linear*, or *right linear*, if no tree in $\Delta_G \setminus \{[S, \varepsilon]\}$ contains a production which is not a Greibach production, a linear production, or a right linear production, respectively. A context-free language L is *linear* or *regular* if there is a linear or right linear context-free grammar G' such that $L = L(G')$, respectively. Otherwise they are called *non-linear* or *non-regular*, respectively.

It is well known that the class of regular languages is a proper subclass of the class of linear languages which is in turn a proper subclass of the class of context-free languages.

In standard textbooks one can find several algorithms to transform context-free grammars such that the resulting grammars have some desirable properties. For instance, each context-free grammar for which $\varepsilon \notin L(G)$ has an equivalent grammar in Greibach normal form. Now each grammar in Greibach normal form is ε-free and it does not have chain productions. Algorithms to eliminate ε-productions and chain productions and to generate a Greibach normal form grammar in case $\varepsilon \notin L(G)$ can be found in [17, Chapter 4]. For $\varepsilon \in L(G)$ we first generate an appropriate grammar $G' = (N', \Sigma', P', S')$ for the language $L(G) \setminus \{\varepsilon\}$. By adding a new nonterminal S'' one easily obtains an appropriate grammar G'' for $L(G)$.[4]

[4] $\{G'' = (N' \cup \{S''\}, \Sigma', P' \cup \{S'' \to \varepsilon\} \cup \{S'' \to \alpha \mid S' \to \alpha \in P'\}, S'')$. Note that it might be the case that S' cannot be generated by G'' anymore.

2.3.4 Properties of Production Sets

There are also some desirable properties which do not concern the form but the cooperation of productions: A symbol $X \in N \cup \Sigma$ is *useful* if there are $\tau_1, \tau_2 \in T_G^*$ and $p \in T_G$ with the property $\ell(p) = X$ and $\tau_1 p \tau_2 \in \Delta_G$. Otherwise X is *useless*. A context-free grammar is *reduced* if it does not contain useless symbols. Useless symbols can easily be detected and eliminated [17, Chapter 4.4]. By definition a symbol is not useless if it occurs in a derivation tree. Hence, the elimination of useless symbols does not affect the set of derivation trees. Since ambiguity is about the structure of the set of derivation trees context-free grammars which are not reduced are not interesting for us. Moreover, for a reduced context-free grammar we have $\mathit{embedded}(\Delta_G) = \Delta_{N \cup \Sigma} \cap T_G^*$, i.e., each tree which can be formed using only terminals, nonterminals and productions is a tree which is embedded in some derivation tree.

The grammar G is *cycle-free* if for all $\rho \in \mathit{embedded}(\Delta_G)$ we have $\uparrow(\rho) = \downarrow(\rho)$ implies $\rho = \uparrow(\rho)$, otherwise it is *cyclic*, and a tree ρ with $\uparrow(\rho) = \downarrow(\rho) \neq \rho$ is a *cyclic tree*. Finally, we call G *proper* if it is cycle-free and reduced. If not stated otherwise throughout this thesis we assume context-free grammars to be proper.

2.3.5 Derivation Trees are Very Simple

Definition 2.53 [5] *A context-free grammar $G = (N, \Sigma, P, S)$ is* very simple *if it is in Greibach normal form and each terminal is generated by a uniquely defined production, i.e., $\forall a \in \Sigma : |P \cap N \times aN^*| = 1$. A language L is very simple if $L = L(G)$ for some very simple grammar G.*

Obviously, each very simple grammar is an $LL(1)$ grammar.[6] Each $LL(1)$ grammar generates a deterministic context-free language which is a well known proper subclass of context-free languages [17].

Lemma 2.54 *For an arbitrary context-free grammar $G = (N, \Sigma, P, S)$ the set of derivation trees Δ_G is a very simple language.*

Proof. Let $\bar{\Sigma} := \{\bar{a} \mid a \in \Sigma\}$ be a new alphabet which is a copy of Σ, i.e., $(N \cup \Sigma) \cap \bar{\Sigma} = \emptyset$ and $|\Sigma| = |\bar{\Sigma}|$. Let $h : (N \cup \Sigma)^* \to (N \cup \bar{\Sigma})^*$ be a homomorphism defined by $h(A) = A$ for $A \in N$ and $h(a) = \bar{a}$ for $a \in \Sigma$.

$$G' := (N \cup \bar{\Sigma},\ \Sigma \cup P,\ P' \cup \{\bar{a} \to a \mid a \in \Sigma\},\ S)$$
$$P' := \{A \to [A, \alpha] h(\alpha) \mid [A, \alpha] \in P\}$$

[5] Very simple grammars are also defined in [2].

[6] $LL(k)$ grammars are common in parsing theory, see [1] for a definition.

Obviously, G' is very simple. To prove $L(G') = \Delta_G$ it suffices to show that the set of sentential forms of G' and the set of sentential derivation trees of G coincide, i.e. we have to show that $h^{-1}(\mathcal{S}_{G'}) = \mathit{cut}(\Delta(G))$. The equation can be divided into two inclusions, each of which can be shown by an elementary induction on the number of internal symbols in a tree belonging to $\mathit{cut}(\Delta_{G'})$ and $\mathit{cut}(\Delta(G))$, respectively. □

By essentially the same technique we obtain:

Lemma 2.55 *For an arbitrary context-free grammar $G = (N, \Sigma, P, S)$ the set of sentential derivation trees $\mathit{cut}(\Delta_G)$ is a very simple language.*

Proof. Let $\bar{\Sigma}$ be defined as in Lemma 2.54. Moreover, let $\bar{N}$ be a copy of N in the same way as $\bar{\Sigma}$ is a copy of Σ. Let $h : (N \cup \Sigma)^* \to (\bar{N} \cup \bar{\Sigma})^*$ be a homomorphism defined by $h(X) = \bar{X}$ for each $X \in N \cup \Sigma$.

$$G'' := (\bar{N} \cup \bar{\Sigma},\, T_G,\, P' \cup \{\bar{X} \to X \mid X \in N \cup \Sigma\},\, \bar{S})$$
$$P' := \{\bar{A} \to [A, \alpha]h(\alpha) \mid [A, \alpha] \in P\}$$

Obviously, G'' is very simple. The proof that $L(G'') = \mathit{cut}(\Delta_G)$ is analogous to the one of Lemma 2.54. □

Lemma 2.54 and Lemma 2.55 show that we can apply known results on very simple languages to the set of derivation trees or sentential derivation trees of any context-free grammar. In particular derivation tree sets and sentential derivation tree sets satisfy each pumping lemma for context-free or deterministic context-free languages.

2.3.6 Parikh Suprema

Definition 2.56 *Let $G = (N, \Sigma, P, S)$ be a context-free grammar. The* Parikh supremum *of G is defined by:* $\sup(G) := \sup(\mathit{cut}(\Delta_G))$*, i.e, for terminals and nonterminals it is the maximal number of occurrences in a sentential form and for productions the maximal number of occurrences in a derivation tree. A symbol $X \in P \cup N \cup \Sigma$ is* bounded *for G if* $\sup(G)(X) \neq \omega$*. Otherwise X is* unbounded *for G. We often call a symbol bounded or unbounded without mentioning the corresponding context-free grammar if it is clear from the context. We call a bounded symbol $X \in P \cup N \cup \Sigma$ a* bounded production*,* bounded nonterminal*, or* bounded terminal*, if X is a production, nonterminal, or terminal, respectively. Analogously we define* unbounded productions*,* unbounded nonterminals*, and* unbounded terminals*. The set of bounded and unbounded productions are denoted by $P_{<\omega}$ and P_ω, respectively.*

2.3.7 Pumping Trees

In this section we show two observations which can be seen as parts of the proof of the well known pumping lemma due to Bar-Hillel [4]. Our aim is not to present an alternative proof for the pumping lemma but to use the more elementary observations later.

Observation 2.57 *Let $\rho \in \Delta_G$ be a derivation tree. If embedded(ρ)$\cap\Lambda_G \neq \emptyset$ then there are $\tau_1, \tau_2, \omega, \tau_4, \tau_5 \in T_G^*$ such that $\rho = \tau_1\tau_2\omega\tau_4\tau_5$, $\omega \in$ subtree(Δ_G), $\tau_2\uparrow(\omega)\tau_4 \in \Lambda_G$, and $\tau_1\tau_2^n\omega\tau_4^n\tau_5 \in \Delta_G$ for each $n \in \mathbb{N}$.*

Proof. If a pumping tree λ is embedded in a derivation tree ρ then by definition λ is obtained from a subtree ρ' of ρ by cutting off a single subtree ω such that $\uparrow(\rho') = \uparrow(\omega)$. Thus, there are $\tau_1, \tau_2, \tau_4, \tau_5 \in T_G^*$ such that $\rho = \tau_1\tau_2\omega\tau_4\tau_5$, $\rho' = \tau_2\omega\tau_4$, and $\lambda = \tau_2\uparrow(\lambda)\tau_4$. It is easily seen by induction on n that $\uparrow(\rho') = \uparrow(\omega)$ permits the iteration $\tau_1\tau_2^n\omega\tau_4^n\tau_5 \in \Delta_G$ for each $n \in \mathbb{N}$. □

Observation 2.58 *For each context-free grammar G the set of derivation trees which does not contain embedded pumping trees is finite, i.e.,*

$$\exists k_G \in \mathbb{N} : |\{\rho \in \Delta_G \mid \textit{embedded}(\rho) \cap \Lambda_G = \emptyset\}| \leq k_G.$$

Proof. According to Observation 2.25 each reducible tree consists of a production followed by a forest. Let $m := \max\{|r(p)| \mid p \in P\}$ be the maximum number of symbols on the right-hand side of a production. One can easily show that each derivation tree whose length is larger than $j := \sum_{i=0}^{|N|+1} m^i = \frac{m^{|N|+2}-1}{m-1}$ has a node with more than $|N|$ many different ancestors. Hence, two of them have the same left-hand side which gives rise to an embedded pumping tree. Clearly the number of derivation trees with length lower than j is bounded by some constant k_G. □

Definition 2.59 *The* pumping constant *c_G of a context-free grammar G is defined by $c_G := |w| + 1$, where $w \in \Sigma^*$ is a longest word such that $w = \downarrow(\rho)$ for some $\rho \in \Delta_G$ not containing embedded pumping trees, i.e., embedded(ρ)$\cap$ $\Lambda_G = \emptyset$. The* pumping constant *c_L of a context-free language L is the least pumping constant which a context-free grammar generating L can have, i.e.,*

$$c_L := \min\{c_G \mid G \textit{ context-free and } L = L(G)\}.$$

Observation 2.58 guarantees that the pumping constants of context-free grammars and languages are well defined positive integers.

Even though this is not their main purpose we sketch how the pumping lemma by Bar-Hillel can be proved from the previous two observations:

Since there is only a finite number of derivation trees which does not contain pumping trees (Observation 2.58) there is a longest word with this property. Each derivation tree ρ for a longer word w has embedded pumping trees. We can erase all the cyclic trees from ρ to obtain a derivation tree ρ' with $\downarrow(\rho') = w$ but without embedded cyclic trees. Now ρ' still contains at least one pumping tree λ which cannot be cyclic. Hence, at least one terminal is generated by λ. By the use of Observation 2.57 the pumping lemma of Bar-Hillel immediately follows.

2.3.8 Types of Productions and Their Properties

In this section $G = (N, \Sigma, P, S)$ is an arbitrary proper context-free grammar, $X, Y \in N \cup \Sigma$, and $a, b, c \in \Sigma$. Definition 2.56 already partitioned the set of productions into two classes, namely bounded and unbounded productions. The following observation shows how they relate to pumping trees.

Observation 2.60 *A production is bounded if and only if it is not contained in any pumping tree, i.e., $p \in P$ is bounded if and only if for each $\vartheta \in \Lambda_G$ we have $p \notin \vec{\vartheta}$.*

Proof. For each pumping tree $\vartheta \in \Lambda_G$ by definition there is a derivation tree ρ such that $\vartheta \in \mathit{embedded}(\rho)$. By Observation 2.57 each production $p \in P$ which is in ϑ (i.e., $p \in \vec{\vartheta}$) cannot be bounded.

Now let us consider a production $p \in P$ such that $p \notin \vec{\vartheta}$ for any pumping tree $\vartheta \in \Lambda_G$. Let $\rho \in \Delta_G$ be an arbitrary derivation tree. According to Observation 2.57 we can erase all the pumping trees of ρ and obtain a derivation tree $\rho' \in \Delta_G$, which does not contain embedded pumping trees. By definition ρ' contains the same number of occurrences of the production p as ρ, that is $|\rho|_p = |\rho'|_p$. By Observation 2.58 the number of derivation trees without embedded pumping trees is finite. Hence, there is one such tree τ with the maximum number k_p of occurrences of the production p, i.e., $k_p = |\tau|_p$. Then $|\rho|_p = |\rho'|_p \leq |\tau|_p = k_p$, and thus p is bounded. Note that k_p only depends on p but not on the choice of ρ. □

Definition 2.61 *Let $\rho \in \Delta \setminus \Gamma$ be a reducible tree then the first symbol $\rho[1]$ is called the* dominating production *of ρ.*

Definition 2.62 *A production $p = [A, \alpha] \in P$ is called a* pumping production *if it is the dominating production of some pumping tree, i.e., $\exists \rho \in \Lambda_G : p = \rho[1]$. Otherwise it is called a* descending production. *The set of pumping productions and descending productions are denoted by $P_=$ and $P_<$, respectively.*

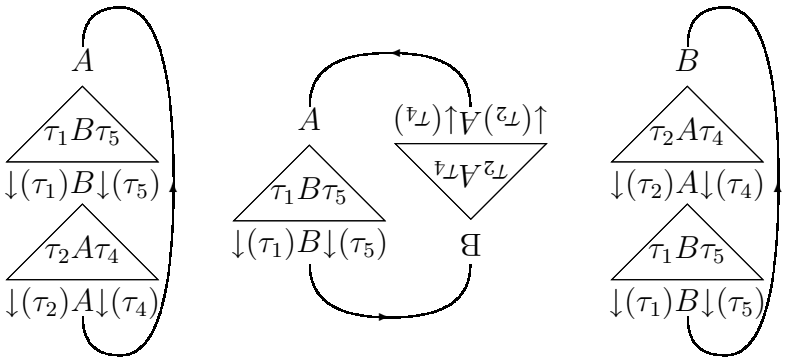


Figure 2.4: Pumping tree rotation.

A pumping production is contained in some pumping tree by definition. Therefore, Observation 2.60 implies that each pumping production is unbounded. Similarly Observation 2.60 implies that bounded productions are descending since they do not occur in any pumping tree. Thus, we partition productions into three disjoint classes: Pumping productions, descending productions which occur in some pumping trees, and bounded productions which do not occur in any pumping tree.

Pumping trees can be rotated in the following sense: The link node and the root node of each pumping tree are labelled with the same symbol. If we identify them we obtain a directed graph which has exactly one cycle with attached trees. We can consider each node on this cycle as a pearl of a necklace. By cutting through an arbitrary pearl we always obtain a pumping tree. This rotation of pumping trees is visualised in Figure 2.4. In the leftmost image of Figure 2.4 an edge between the link node and the root is added to obtain a cycle. Then we duplicate some node (labelled with B) between the link node and the root node. To keep the cycle we connect the duplicates by an edge. This situation is depicted in the middle. Finally, we identify the former link node and root node and remove the edge between them. This leads to the rightmost image. By removing the new edge we obtain a rotated version of the original pumping tree. Clearly we could have avoided to introduce the new edge which is only added to keep the picture of pearls moving around a necklace. The pumping tree rotation is handled in the following observation formally:

Observation 2.63 *Let $\tau_1, \tau_2, \tau_4, \tau_5 \in T_G^*$ and $A \in N$ such that $\tau_1\tau_2 A\tau_4\tau_5 \in \Lambda_G$ and $\tau_2 A\tau_4 \in$ embedded(Δ_G). Then we have $\tau_2\tau_1 B\tau_5\tau_4 \in \Lambda_G$, where $B := \uparrow(\tau_2 A\tau_4)$.*

Proof. Let $\tau_1, \tau_2, \tau_4, \tau_5 \in T_G^*$ and $A \in N$ such that $\tau_1\tau_2 A\tau_4\tau_5 \in \Lambda_G$ and

$\tau_2 A \tau_4 \in \mathit{embedded}(\Delta_G)$. Moreover, let $B := \uparrow(\tau_2 A \tau_4)$. Then by Observation 2.16 we obtain:

$$A = \uparrow(\tau_1\tau_2 A \tau_4\tau_5) = \uparrow(\tau_1\uparrow(\tau_2 A\tau_4)\tau_5) = \uparrow(\tau_1 B\tau_5). \qquad (i)$$

$$\text{(i)} \Rightarrow \quad \uparrow(\tau_2\tau_1 B\tau_5\tau_4) = \uparrow(\tau_2\uparrow(\tau_1 B\tau_5)\tau_4) = \uparrow(\tau_2 A\tau_4) = B.$$

Thus, $\tau_2\tau_1 B\tau_5\tau_4$ has only one nonterminal in the frontier and this one coincides with the root. Since G is reduced, any tree which can be generated with the letters in P_G^* is an embedded tree. Therefore, we obtain $\tau_2\tau_1 B\tau_5\tau_4 \in \Lambda_G$. □

Observation 2.64 *For each pumping tree each proper ancestor of the link node has a pumping production as its parse label. For each descending production p which is unbounded there is a pumping tree such that p is the parse label of some node μ such that the first common ancestor of μ and the link node is the root node.*

Proof. Due to the pumping tree rotations (Observation 2.63) each ancestor of the link node has a pumping production as its parse label. According to Observation 2.60 the production p is contained in some pumping tree λ. Let μ be a node of λ with the parse label p, i.e., $\lambda[\mu] = p$. Let ν be the first common ancestor of μ and the link node. Then according to Observation 2.63 we can rotate λ such that ν is moved to the top of the tree. □

Type Computation

We have divided the set of productions into three types. Namely pumping productions, bounded productions, and unbounded descending productions. The type of a production is a property which is not local in the sense that it often cannot be decided without knowledge about other productions. In contrast to that, the decision whether a production is linear or in Greibach form only requires to consider the production itself. The non-local aspects required to decide the non-local properties above are covered in the so called condensation of the dependency graph, which can be viewed as a partial order on an equivalence relation on the set of terminals and nonterminals. It turns out that this partial order can be computed efficiently with respect to the size of the context-free grammar G.

Definition 2.65 *We say X is a* potential ancestor *of Y, denoted by $X \vdash Y$, if there exists a derivation tree $\rho \in \Delta_G$ and $i, j \in [1, |\rho|]$ such that i is an ancestor of j in ρ and $\ell(\rho[i]) = X$ and $\ell(\rho[j]) = Y$. The set of potential ancestors of Y is denoted by ∇_Y, i.e., $\nabla_Y := \{A \in N \cup \Sigma \mid A \vdash Y\}$.*

For a reduced context-free grammar the relation $\vdash$ can be computed easily as the reflexive and transitive closure of the relation

$$\vdash_1 := \{(A, X) \in N \times (N \cup \Sigma) \mid \exists [A, \alpha] \in P : X \in \vec{\alpha}\}.$$

The relation $\vdash_1$ can be read from P directly. If we consider the terminals and nonterminals as nodes and the pairs of $\vdash_1$ as edges of a directed graph we obtain the well known *dependency graph.*[7]

Lemma 2.66 *A production $p = [A, \alpha] \in P$ is a pumping production if and only if its right-hand side contains a symbol which is a potential ancestor of the symbol on the left-hand side, i.e., $\exists B \in \vec{\alpha} : B \vdash A$.*

Proof. Let $p = [A, \alpha] \in P$. Firstly we consider the case $B \vdash A$ for some $B \in \vec{\alpha}$. Since G is reduced there is a $\rho \in \mathit{subtree}(\Delta_G)$ such that p is the dominating production. Then one child of the root node of ρ is the root node of a tree $\rho' \in \mathit{subtree}(\rho)$ such that $\uparrow(\rho') = B$. Hence, $\rho = p\tau_1\rho'\tau_5$ for some $\tau_1, \tau_5 \in (P \cup \Sigma)^*$. Since B is a potential ancestor of A there is a tree $\tau \in \mathit{subtree}(\Delta_G)$ which has a subtree $\tau' \in \mathit{subtree}(\tau)$ such that $\uparrow(\tau) = B$ and $\uparrow(\tau') = A$. Thus, for some $\tau_2, \tau_4 \in (P \cup \Sigma)^*$ we have $\tau = \tau_2\tau'\tau_4$. According to Lemma 2.16 and by the fact that G is reduced we can write

$$\begin{aligned} A = \uparrow(\rho) = \uparrow(p\tau_1\rho'\tau_5) = \uparrow(p\tau_1\uparrow(\rho')\tau_5) = \uparrow(p\tau_1 B\tau_5) = \uparrow(p\tau_1\tau\tau_5) \\ = \uparrow(p\tau_1\tau_2\tau'\tau_4\tau_5) = \uparrow(p\tau_1\tau_2\uparrow(\tau')\tau_4\tau_5) = \uparrow(p\tau_1\tau_2 A\tau_4\tau_5) \end{aligned}$$

Moreover, $\downarrow(p\tau_1\tau_2 A\tau_4\tau_5) \in \Sigma^* A \Sigma^*$. Hence, $p\tau_1\tau_2 A\tau_4\tau_5 \in \Lambda_G$ and p is a pumping production.

Secondly we consider the case $\forall B \in \vec{\alpha} : \neg(B \vdash A)$. Then for a $\rho \in \mathit{subtree}(\Delta_G)$ such that p is the dominating production there can be no node except the root node which can be labelled by A. Hence, there is no pumping tree which is dominated by p. Hence, p is not a pumping production. □

Using Lemma 2.66 we can easily provide an efficient algorithm to detect whether a production p is a pumping production:

(i) Compute the relation $\vdash_1$.

(ii) Compute $\vdash$, which is the reflexive and transitive closure of $\vdash_1$.

(iii) Examine whether there is a nonterminal B on the right-hand side of p which is a potential ancestor of the symbol $\ell(p)$ on the left-hand side, i.e., whether $B \vdash \ell(p)$

[7] Note that in the literature often only nonterminals are considered as nodes of the dependency graph.

Due to Lemma 2.66 this algorithm is correct.

Now we know how to find out whether a production is a pumping or descending production and we know that pumping productions are not bounded. It remains to distinguish unbounded and bounded descending productions. To check whether a production is bounded we have to find out whether or not it can occur in any pumping tree. Since the set of pumping trees is either empty or infinite we cannot expect to examine all the pumping trees directly. However, it is possible to construct and inspect the finite number of cuts of pumping trees which have the root in their frontier and which do not contain other embedded trees with this property. It can be shown that it is sufficient to consider these trees. But their number may be exponential with respect to the size of the grammar. Fortunately we can do better than this. We start our development of a better algorithm by observating some basic properties of descending productions:

Observation 2.67 *If a descending production p is the dominating production of some subtree of a derivation tree $\rho \in subtree(\Delta_G)$ then the root node is the only node labelled by $\ell(p)$, i.e., $\forall i \in [2, |\rho|] : \rho[i] \neq p$.*

Proof. Let $p \in P$ dominate a tree $\rho \in subtree(\Delta_G)$ and ρ has a node $\mu \neq 1$ such that $\rho[\mu] = p$. Then by Observation 2.36 we have $\rho = p\tau_1 p\tau_2\tau_3$ for some $\tau_1, \tau_2, \tau_3 \in (P \cup \Sigma)^*$ such that $p\tau_2 \in subtree(\rho)$. But then cutting off the tree $p\tau_2$ at the node $|p\tau_1| + 1$ yields the pumping tree $p\tau_1\ell(p)\tau_3 \in \Lambda_G$. Thus, p is the dominating production of a pumping tree and therefore a pumping production. Hence, a descending production p cannot be the dominating production of a subtree $\rho' \in subtree(\Delta_G)$ which has a node $\mu \neq 1$ such that $\rho[\mu] = p$. In other words $\forall i \in [2, |\rho|] : \rho[i] \neq p$. □

Observation 2.68 *Let $p \in P$ be a bounded production and $A := \ell(p)$. Then $\sup(\mathcal{S}_G)(A) = \sup(\Delta_G)(p)$, i.e., the maximum number of A's which can occur in a sentential form equals the maximum number of occurrences of the production p in any derivation tree.*

Proof. Let $\alpha \in \mathcal{S}_G$ be an arbitrary sentential form. Then α is the frontier of some sentential derivation tree $\rho \in cut(\Delta_G)$. One can expand each occurrence of an A in the frontier by the production p. Since G is reduced the resulting tree ρ' is also a cut of a derivation tree, i.e., $\rho' \in cut(\Delta_G)$. Hence, there is a derivation tree $\rho'' \in \Delta_G$ such that $\rho' \in cut(\rho'')$. Then ρ'' has at least $|\rho|_A$ many occurrences of p, i.e. ,$\sup(\mathcal{S}_G)(A) \leq \sup(\Delta_G)(p)$.

On the other hand let $\tau_1\omega\tau_2 \in embedded(\Delta_G)$ such that $p = \omega[1]$. Note that $A = \ell(p)$. By Observation 2.67 we know that ω does not contain

any occurrence of p except the dominating one, i.e. $|\omega|_p = 1$. Therefore, $|\tau_1\omega\tau_2|_p = |\tau_1\uparrow(\omega)\tau_2|_p + 1$. Again by Observation 2.67 we obtain that $|\tau_1\omega\tau_2|_A = |\tau_1\uparrow(\omega)\tau_2|_A - 1$. Hence, $|\tau_1\omega\tau_2|_F = |\tau_1\uparrow(\omega)\tau_2|_F$ for $F := \{A, p\}$. A derivation tree $\rho \in \Delta_G$ has no nonterminal in the frontier, i.e., $|\rho|_N = 0$. By the invariance shown above cutting off all the subtrees dominated by p yields a tree $\rho' \in \mathit{cut}(\Delta_G)$ such that $|\rho'|_A = |\rho|_p$. Hence, $\sup(\mathcal{S}_G))(A) \geq \sup(\Delta_G)(p)$. This completes the proof. □

As an immediate consequence of Observation 2.68 and Observation 2.60 we obtain:

Lemma 2.69 *A production p is bounded if and only if it is descending and its left-hand side $\ell(p)$ is bounded.*

We have already seen an efficient algorithm to detect whether a production is a pumping or a descending production. According to Lemma 2.69 we can efficiently detect whether a production is bounded provided we have an efficient algorithm detecting bounded nonterminals. Thus, we now examine when a symbol is bounded.

It is not the case that each $X \in N \cup \Sigma$ which is the label of a node in a pumping tree is unbounded. Note that only nodes whose parse label is X are relevant for $\sup(\mathcal{S}_G)(X)$, i.e., only leaves labelled X are relevant. For instance consider the context-free grammar:

$$S \to aSb \mid \varepsilon$$

In the pumping tree $\vartheta := [S, aSb]aSb$ we find the tree symbols a, b, and S. But S obviously occurs only at most once in any sentential form, while a and b are unbounded. An iteration of ϑ always consumes and reproduces an S. In this sense the link node S is the pump itself and the symbols a and b are the pumped objects. Pumping increases the number of pumped objects but not necessarily the number of pumps. A symbol is unbounded if and only if it is the label of some node in a pumping tree which is not an ancestor of the link node. Then pumping leads to independent copies of such a node. In the sequel we develop an algorithm to decide this property efficiently. We start with the definition of an equivalence relation on the nonterminals and terminals.

Definition 2.70 *We say that X and Y are equivalent denoted by $X \equiv Y$ if $X \vdash Y$ and $Y \vdash X$. For each $X \in N \cup \Sigma$ the corresponding equivalence class is denoted by:*

$$[X] := \{Y \in N \cup \Sigma \mid X \equiv Y\}.$$

Since $\vdash$ is reflexive and transitive so is $\vdash^{-1}$. The intersection of relations which are reflexive and transitive yield a relation which again is reflexive and transitive. Hence, $\equiv = \vdash \cap \vdash^{-1}$ is reflexive and transitive. Finally, $\equiv$ is symmetric by definition. Thus, we have verified that $\equiv$ is in fact an equivalence relation. Note that the equivalence classes correspond to strong components of the dependency graph, and terminals form singleton equivalence class. It is well known that strong components can be computed in linear time with respect to the size (the sum of nodes and edges) of a graph.

Since each symbol B on the right-hand side of a production p has the left-hand side as a potential ancestor, i.e. $\ell(p) \vdash B$, we can restate Lemma 2.66 in the following way:

Lemma 2.71 *A production $p = [A, \alpha] \in P$ is a pumping production if and only if its right-hand side contains a symbol which is in the same equivalence class as the left-hand side, i.e., $\exists B \in \vec{\alpha} : [A] = [B]$.*

Lemma 2.71 shows that to compute whether a production is a pumping production or not it is sufficient to precompute the equivalence classes.

This is just a first application of the equivalence relation $\equiv$. More importantly this relation will turn out to be crucial to prove an easily checkable necessary and sufficient criterion for bounded productions.

Definition 2.72 *The set of equivalence classes is defined by:*

$$\mathcal{N}_G := \{[X] \mid X \in N \cup \Sigma\}$$

On $\mathcal{N}_G$ the relation $\vdash$ is defined by $[X] \vdash [Y]$ if $X \vdash Y$.

Observation 2.73 *The relation $\vdash$ on $\mathcal{N}_G$ is a well defined partial order.*

Proof. To see that $\vdash$ is well defined on $\mathcal{N}_G$ it suffices to show that for each $X' \in [X]$ and each $Y' \in [Y]$ we have $X \vdash Y \Rightarrow X' \vdash Y'$. Since $X' \in [X]$ we have $X' \vdash X$ similarly $Y' \in [Y]$ implies $Y \vdash Y'$. Thus, if $X \vdash Y$ holds we have $X' \vdash X \vdash Y \vdash Y'$. By the transitivity of $\vdash$ this implies $X' \vdash Y'$. Hence, $\vdash$ is well defined on $\mathcal{N}_G$. Clearly $\vdash$ is reflexive and transitive on $\mathcal{N}_G$. Moreover, by definition $[A] \vdash [B]$ and $[B] \vdash [A]$ implies $[A] = [B]$. Hence, $\vdash$ is also antisymmetric on $\mathcal{A}$. Therefore, $\vdash$ is a partial order on $\mathcal{N}_G$. □

Again we can consider the relation $\vdash$ on equivalence classes as edges of a directed graph. The nodes of this graph G are the equivalence classes. The graph G is called the *condensation* of the dependency graph, i.e., we have condensed each strong component of the dependency graph into a single node.

Now we define the distance of two symbols X and Y as the longest path in the condensation graph from $[X]$ to $[Y]$. Formally that is:

Definition 2.74 *Let $X \in N \cup \Sigma$ and $Y \in \nabla_X$. Then the* distance *from Y to X is defined by:*

$$\begin{aligned} dist(Y,X) \;:=\; & \max\big\{i \in \mathbb{N} \mid \exists Z_0, \dots Z_i \in \mathcal{N}_G : \forall j \in [1,i] : \\ & Z_{j-1} \vdash Z_j \text{ and } Z_{j-1} \not\equiv Z_j \text{ and } Y \in Z_0 \text{ and } X \in Z_i\big\}. \end{aligned}$$

Definition 2.75 *Let $X, Y \in N \cup \Sigma$ be symbols and let*

$$k_X := \max\{|r(p)|_{\nabla_X} \mid p \in P\}.$$

Thus, k_X is the maximal number of symbols on the right-hand side of a production which are potential ancestors of X. The X-weight of Y is

$$w_X(Y) := \begin{cases} 0 & \text{if } Y \notin \nabla_X \\ k_X^{dist(Y,X)} & \text{otherwise.} \end{cases}$$

We extend w_X to a homomorphism $w_X : (N \cup \Sigma)^* \to \mathbb{N}$.

Definition 2.76 *A symbol $X \in N \cup \Sigma$ is* pumpable *if there is a pumping production p such that at least two symbols on the right-hand side of p are in ∇_X, i.e., $|r(p)|_{\nabla_X} > 1$. A production p with this property is called a* witness of the pumpability *of X.*

Lemma 2.77 *Let $p \in P$ be a production which is not a witness for the pumpability of X. Then the X-weight of the left-hand side of p is at least as large as the X-weight of its right-hand side, i.e.,*

$$w_X(\ell(p)) \geq w_X(r(p)).$$

Proof.

Case 1: $\ell(p) \notin \nabla_X$. In this case no symbol on the right-hand side of p is in ∇_X. Hence, $w_X(r(p)) = w_X(\ell(p)) = 0$.

Case 2: $\ell(p) \in \nabla_X$

Case 2.1: p **is descending.** In this case all the symbols on the right-hand side of p which are in ∇_X, which are at most k_X many, have a strictly lower distance to X than $\ell(p)$. Hence, for $z := \pi_{\nabla_X}(r(p))$ we have:

$$\begin{aligned} w_X(\ell(p)) &= k_X^{dist(\ell(p),X)} = k_X \cdot k_X^{dist(\ell(p),X)-1} \\ &\geq \sum_{A \in \vec{z}} k_X^{dist(\ell(p),X)-1} \geq \sum_{A \in \vec{z}} k_X^{dist(A,X)} \quad \geq w_X(z) \\ &= w_X(r(p)). \end{aligned}$$

Case 2.2: p is pumping but $|r(p)|_{\nabla_X} \leq 1$. Let $\alpha := r(p)$. Since p is a pumping production there is a nonterminal $B \in \vec{\alpha} \cap [\ell(p)]$. Then $B \in \nabla_X$ and $dist(\ell(p), X) = dist(B, X)$. In addition, $|r(p)|_{\nabla_X} \leq 1$ implies that there is only one occurrence of B in $\vec{\alpha}$ and all the other symbols in $\vec{\alpha}$ have an X-weight of 0. Hence:

$$w_X(\ell(p)) = w_X(B) = w_X(r(p)).$$

There is no other case such that p is not a witness for the pumpability of X. □

Lemma 2.78 *A symbol X is pumpable if and only if it is not bounded.*

Proof. If X is pumpable then there is a production $p \in P$ witnessing this property. Now we take the embedded tree $\rho_o := pr(p)$ which has only one internal node. On the right-hand side of p, i.e., at the frontier of ρ_0, we find an occurrence of a nonterminal Y which is a potential ancestor of X, i.e., there is a tree $\rho_1 \in embedded(\Delta_G)$ such that $\uparrow(\rho_1) = Y$ and $X \in \vec{\rho_1}$. Moreover, there is a *different* occurrence of a nonterminal B in the right-hand side of p, i.e., a *different* leaf at the frontier of ρ_0, such that B is a potential ancestor of $\ell(p)$, i.e., the root of ρ_0. Therefore, there is a tree $\rho_2 \in embedded(\Delta_G)$ such that $B = \uparrow(\rho_2)$ and $\ell(p) \in \vec{\rho_2}$. That means that we can attach ρ_1 and ρ_2 to different nodes of ρ_0. Let $\rho \in embedded(\Delta_G)$ be the result of this operation. Now we have as well $\ell(p) = \uparrow(\rho)$ and X in the frontier. Since X may be a nonterminal, ρ may not be a pumping tree, but clearly we can iterate ρ in the same way we did in Observation 2.57 resulting in arbitrary many copies of X. Since G is reduced, each iterated version of ρ can be found embedded in some derivation tree.

If X is not pumpable then none of the productions is a witness for the pumpability of X. Thus, by Lemma 2.77 the X-weights of sentential forms cannot increase during a derivation. Hence, there can be no sentential form with an X-weight higher than $w_X(S)$. Since the X-weight of X is 1 we obtain that $w_X(S)$ is an upper bound for the number of X occurrences in a sentential form. □

In the sequel we will use the notion pumpable symbol as a synonym for non-bounded symbols, without explicitly referencing the previous lemma.

Definition 2.79 *A context-free grammar $G = (N, \Sigma, P, S)$ is called* nonterminal bounded *if there is a constant $k \in \mathbb{N}$ such that no sentential form contains more than k nonterminals, i.e., $\forall \rho \in cut(\Delta_G) : |\pi_N(\rho)| \leq k$. If G is nonterminal bounded then the least upper bound for the number of nonterminals which can appear in a sentential form $(\max\{|\pi_N(\rho)| \mid \rho \in cut(\Delta_G)\})$ is called the* width *of G.*

At first glance it is not completely obvious how to check whether a context-free grammar is nonterminal bounded or not. The following Lemma provides a necessary and sufficient criterion which is easily checkable.

Lemma 2.80 *A context-free grammar $G = (N, \Sigma, P, S)$ is nonterminal bounded if and only if each pumping production is linear.*

Proof. Let G be an arbitrary context-free grammar such that each pumping production is linear. Since terminals are never potential ancestors of nonterminals the right-hand side of a linear production contains at most one potential ancestor of an arbitrary nonterminal. Hence, no linear production can ever be a witness for the pumpability of any nonterminal. This implies that no nonterminal is pumpable. According to Lemma 2.78 this means that each nonterminal is bounded. Therefore, G is nonterminal bounded.

On the other hand, a non-linear pumping production p contains two distinct occurences of nonterminals A and B such that $A \in [\ell(p)]$.[8] Since $B \in \nabla_B$ this means that B is pumpable and according to Lemma 2.78 not bounded. Hence, G cannot be nonterminal bounded. □

Nonterminal bounded grammars can be seen as a generalisation of linear context-free grammars. A linear context-free grammar is obviously nonterminal bounded with width 1. In the literature one also finds the definition of ultralinear grammars. They coincide with nonterminal bounded grammars [16, page 184, Exercise 3]. The definition of ultralinear grammars is based on the existence of a partition of the nonterminals in a linear ordered hierarchy of equivalance classes. A symbol on the right-hand side of a production cannot be higher in the hierarchy than the symbol on the left-hand side, and if it is on the same level then the production is linear [16, Section 5.7, page 181]. This definition resembles the criterion for nonterminal bounded grammars in Lemma 2.80, but it does not provide a hint which hierarchy could be useful. Therefore, it is more obvious how to check our criterion since it is basesd on the fixed partial order induced by $\vdash$ on the equivalence classes of nonterminals.

When we discuss the results of Earley's work [11] in chapter 5 we will come across metalinear grammars which lie between linear and nonterminal bounded grammars, therefore we define them here:

Definition 2.81 *A context-free grammar $G = (N, \Sigma, P, S)$ is* metalinear *if the start symbol does not occur on the right-hand side of any production and*

[8] Note that the occurences of A and B are distinct. This does not imply $A \neq B$.

each production with a left-hand side other than the start symbol is linear, i.e., for $N' := N \setminus \{S\}$ *we have:*

$$P \subseteq (\{S\} \times (N' \cup \Sigma)^*) \cup (N' \times \Sigma^*(N' \cup \{\varepsilon\})\Sigma^*).$$

A language is metalinear *if it is generated by some metalinear grammar.*

It is easily seen that a language is metalinear if and only if it lies in the closure of linear context-free languages under union and concatenation.

2.4 Asymptotic Notations

Let $\mathbb{R}_+$ denote the set of positive real numbers.

Definition 2.82 *Let* $f : \mathbb{N} \to \mathbb{R}_+$ *be a total function. We define:*

- $\mathcal{O}(f) := \{g : \mathbb{N} \to \mathbb{R}_+ \mid \exists c \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N} : \forall n > n_0 : g(n) \leq c \cdot f(n)\}$.
- $\Omega(f) := \{g : \mathbb{N} \to \mathbb{R}_+ \mid f \in \mathcal{O}(g)\}$.
- $\Theta(f) := \mathcal{O}(f) \cap \Omega(f)$.
- $\omega(f) := \Omega(f) \setminus \Theta(f)$.[9]
- $o(f) := \mathcal{O}(f) \setminus \Theta(f)$.

The reader is assumed to be familiar with these notions.

When we draw the graph of a function in such a way that the domain is depicted horizontally and the range vertically then a function g is in $\mathcal{O}(f)$ if a vertical stretch of f by a constant factor c is sufficient to exceed the function g for all but a finite number of arguments. For our application a horizontal stretch turns out to be more natural, which motivates the following definition:

Definition 2.83 *Let* $f : \mathbb{N} \to \mathbb{R}_+$ *be a total function. We define:*

- $\mathcal{O}^T(f) := \{g : \mathbb{N} \to \mathbb{R}_+ \mid \exists c \in \mathbb{N} : \exists n_0 \in \mathbb{N} : \forall n > n_0 : g(n) \leq f(c \cdot n)\}$.
- $\Omega^T(f) := \{g : \mathbb{N} \to \mathbb{R}_+ \mid f \in \mathcal{O}^T(g)\}$.
- $\Theta^T(f) := \mathcal{O}^T(f) \cap \Omega^T(f)$.

[9]Note that we use the Greek letter ω also in other contexts where it means a tree string or a non-bounded component of a Parikh vector. The respective meaning is always clear from the context.

- $\omega^T(f) := \Omega^T(f) \setminus \Theta^T(f)$.
- $o^T(f) := \mathcal{O}^T(f) \setminus \Theta(f)$.

The superscript T is meant to remind to the transposition operation for matrices. There the role of rows and columns are swapped. Similarly here we swap the vertical stretch by a horizontal one.

Compared to the notations in Definition 2.82, the notations of Definition 2.83 turn out to be finer with respect to functions which grow slowly, while it is the other way round for functions which grow fast.

Example 2.84

- $\Theta^T(1) \neq \Theta^T(2)$ *while* $\Theta(1) = \Theta(2)$.
- $\Theta^T(\log n) \neq \Theta^T(\log(n^2))$ *while* $\Theta(\log n) = \Theta(\log(n^2))$
- $\forall k \in \mathbb{R}_+ : \Theta(n^k) = \Theta^T(n^k)$
- $\Theta^T(2^n) = \Theta^T(2^{2n})$ *while* $\Theta(2^n) \neq \Theta(2^{2n})$

A frequent outcome of ambiguity investigations for a context-free language L is, that each context-free grammar generating L has at least $k \in \mathbb{N}$ derivation trees for some word. But the length of the shortest word with at least k derivation trees can only be specified up to a constant factor depending on the chosen context-free grammar G. Usually this constant factor is the pumping constant of G. On the other hand for a fixed length n we can always find a context-free grammar generating all the words up to length n with only one derivation tree. Roughly speaking, sometimes it is easy to determine the length of the shortest word with a fixed ambiguity, up to a constant factor, while it is impossible to prove ambiguity for words of fixed length. Therefore, the transposed versions of the asymptotic notations turn out to be more suitable for ambiguity considerations.

2.5 Ambiguity

We can regard trees as circuits with the root as an input line and the leaves as output lines. The tree expansion allows to "plug" the root A of a tree into a leaf of another, provided that the leaf is of the "appropriate type", i.e., it is labelled with A. For the question how trees can be connected only the roots and frontiers are relevant, while the internal structure can be considered as a black box. Thus, if we want to know how a tree can be connected with

other trees we can avoid to consider its internal structure and concentrate on its interface, which is described by a pair consisting of the root and the frontier. Note that the root of a tree over a finite alphabet Γ is an element of Γ and the frontier is an element of Γ^*. Hence, an interface is an element of P_Γ. From this point of view we can consider the production set of a context-free grammar as the set of interfaces of atomic embedded trees. Formally we define the interface of a tree as follows:

Definition 2.85 *The* interface *of a tree string $\rho \in T^*_\Gamma$ is the pair*

$$\updownarrow_\Gamma(\rho) := [\uparrow(\rho), \downarrow(\rho)].$$

The subscript Γ of $\updownarrow_\Gamma$ is dropped in the sequel because it is always implicitly given by the strings on which we apply this function. Throughout Section 2.5 we assume that Γ is a finite leaf alphabet.

If a set of trees L contains two trees with the same interface i then i does not uniquely specify a tree of L. In this case we call i ambiguous in L. This is a generalisation of the classical ambiguity notion for context-free grammars to arbitrary tree sets. What is the use of this generalisation? It allows to decompose the set of derivation trees into subsets, which can be analysed separately for their ambiguity. We will exploit this strategy in 7.4 extensively. Sometimes it also allows to express facts very intuitively and succinctly. For instance, later it will be shown that a grammar is exponentially ambiguous if and only if its subset of pumping trees is ambiguous.

2.5.1 Ambiguity of Tree Languages

In the sequel let $L \subseteq \Delta_\Gamma$ over a finite alphabet Γ. The ambiguity of an interface $i \in P_\Gamma$ is infinite if there are infinitely many trees in L with the interface i. Otherwise the ambiguity of i is the number of trees in L with the interface i.

Definition 2.86 *The* ambiguity series *of L is the function $d_L : P_\Gamma \to \mathbb{N} \cup \omega$ defined by:*

$$d_L(i) := |\{\tau \in L \mid i = \updownarrow(\tau)\}|$$

The ambiguity of an interface $i \in P_\Gamma$ is $d_L(i)$.

The ambiguity series of a tree language maps a natural number n to the ambiguity of the most ambiguous interface whose right-hand side has length at most n.

Definition 2.87 *The* ambiguity function *of L is the function $\hat{d}_L : \mathbb{N} \to \mathbb{N} \cup \omega$ defined by:*

$$\hat{d}_L(n) := \max\{d_L(i) \mid i \in P_\Gamma \text{ and } r(i) \in \Gamma^{\leq n}\}$$

A tree language $L \subseteq \Delta_\Gamma$ is unambiguous *if $d_L(i) \leq 1$ for all $i \in P_\Gamma$, it is* ambiguous *otherwise.*

By definition $\hat{d}_L$ is a non-decreasing function.

Definition 2.88 *We define the predicate U on tree languages as follows:*

$$\forall L \subseteq \Delta_\Gamma : U(L) :\Leftrightarrow L \text{ is unambiguous.}$$

In Chapter 7 this definition turns out to be particularly useful. It allows to analyse the ambiguity of subsets of derivation trees or embedded trees of a context-free grammar. It also helps to explain results in a succinct way. For instance we will find out that a cycle-free context-free grammar is exponentially ambiguous if and only if its set of pumping trees is ambiguous.

2.5.2 Ambiguity of Context-Free Grammars

Throughout Section 2.5.2 let $G = (N, \Sigma, P, S)$ be a context free grammar.

Usually the ambiguity of a word $w \in \Sigma^*$ generated by a context-free grammar $G = (N, \Sigma, P, S)$ is defined as the number of derivation trees with the frontier w. Since the root of each derivation tree is labelled S, we can also consider the ambiguity of w as the number of derivation trees with interface $[S, w]$. Moreover, we consider the set of G's derivation trees as a formal language over $T_{N \cup \Sigma}$. Thus, we can generalise the notion of ambiguity from words to interfaces and from derivation tree sets to arbitrary formal languages over a tree alphabet.

Definition 2.89 *The* interface power series *of G is the function $d_G := d_{cut(\Delta_G)}$, i.e., the interface ambiguity of G is the interface ambiguity of the set of G's embedded trees.*

We also use the name d_G for the function where the root component of the interface is fixed to be the start-symbol. Which function is meant is clear from the type of argument. This technique is known as overloading in some programming languages.

Definition 2.90 *The* ambiguity power series *is the mapping $d_G : (N \cup \Sigma)^* \to \mathbb{N} \cup \omega$ defined by $d_G(\alpha) := d_G([S, \alpha])$.*

Mappings like d_G are presented as formal sums in several textbooks. For instance we can write d as $\sum_{w\in\Sigma^*} d(w)w$. This habit inspires the name "ambiguity power series" even though we seldom use formal sums in this thesis.

Definition 2.91 *The* ambiguity function of G *is the mapping* $\hat{d}_G := \hat{d}_{\Delta_G}$, *i.e., the ambiguity function of* G *is the interface ambiguity function of* G*'s derivation trees. We say that* G *is* $\hat{d}_G$-ambiguous.

The ambiguity function $\hat{d} : \mathbb{N} \to \mathbb{N}$ maps each n in $\mathbb{N}$ to the ambiguity of the most ambiguous word of length up to n. Note that by definition the ambiguity of sentential forms containing nonterminals is not taken into account for the ambiguity function.

Definition 2.92 *Let* $f : \mathbb{N} \to \mathbb{N}$ *be a function. The grammar* G *is* $\mathcal{S}(f)$-ambiguous *if* $\hat{d}_G \in \mathcal{S}(f)$, *where* $\mathcal{S}(f)$ *is one of the sets* $\mathcal{O}(f)$, $\mathcal{O}^T(f)$, $o(f)$, $o^T(f)$, $\Omega(f)$, $\Omega^T(f)$, $\omega(f)$, $\omega^T(f)$,$\Theta(f)$, *or* $\Theta^T(f)$.

Definition 2.93 *A context-free grammar* G

- *is* k-ambiguous *or* ambiguous of degree k *for a* $k \in \mathbb{N}$ *if* $\hat{d}_G \in \Theta^T(k)$, *i.e., if* $\hat{d}_G$ *is bounded by* k *but not by* $k-1$.
- *is* unambiguous *if it is* 1*-ambiguous or* 0*-ambiguous otherwise it is* ambiguous. *The class of unambiguous context-free grammars is denoted by UCFG.*
- *has* finite ambiguity *or* finite degree of ambiguity *if it is* $\mathcal{O}(1)$*-ambiguous. Otherwise we say that* G *has* infinite ambiguity *or* infinite degree of ambiguity. *The class of context-free grammars with finite degree of ambiguity is denoted by FCFG.*
- *has* polynomially bounded ambiguity *if* $\hat{d}_G \in \mathcal{O}(n^k)$ *for some* $k \in \mathbb{N}$. *The class of context-free grammars with polynomially bounded ambiguity is denoted by PCFG.*
- *is* exponentially ambiguous *if* $\hat{d}_G \in \Theta^T(2^n)$. *The class of context-free grammars with* exponential ambiguity *is denoted by ECFG.*

Note that a context-free grammar is 0-ambiguous if and only if $L(G) = \emptyset$.

As indicated by Example 2.84 the superscript T in the definition of *ECFG* is essential. In contrast to that the sets $\mathcal{O}(n^k)$ and $\mathcal{O}^T(n^k)$ coincide. Thus, in the definition of *PCFG*, we could have replaced $\mathcal{O}$ by $\mathcal{O}^T$.

2.5.3 Ambiguity of Context-Free Languages

Definition 2.94 *Let $f : \mathbb{N} \to \mathbb{R}_+$ be a monotone function. A context-free language L is:*

- $\mathcal{O}(f)$-, $\mathcal{O}^T(f)$-, $\mathrm{o}(f)$-, *or* $\mathrm{o}^T(f)$-ambiguous *if it is generated by a context-free grammar which is $\mathcal{O}(f)$-, $\mathcal{O}^T(f)$-, $o(f)$-, or $o^T(f)$-ambiguous, respectively.*
- $\Omega(f)$-, $\Omega^T(f)$-, $\omega(f)$-, *or* $\omega^T(f)$-ambiguous *if it is only generated by context-free grammars which are $\Omega(f)$-, $\Omega^T(f)$-, $\omega(f)$-, or $\omega^T(f)$-ambiguous.*
- $\Theta(f)$-ambiguous *or* $\Theta^T(f)$-ambiguous *if it is $\mathcal{O}(f)$- and $\Omega(f)$-ambiguous, or $\mathcal{O}^T(f)$- and $\Omega^T(f)$-ambiguous, respectively.*

In contrast to the $\mathcal{O}^T$ and Ω^T notations the $\mathcal{O}$ and Ω notations are very rough for low ambiguities and at the same time too precise for exponential ambiguity. For example a context-free grammar is $\Theta(1)$-ambiguous if and only if it has finite degree of ambiguity. To distinguish different degrees of ambiguity we need the Θ^T notation. As defined above a context-free grammar is k-ambiguous for a $k \in \mathbb{N}$ if and only if it is $\Theta^T(k)$-ambiguous. Moreover, we will see later that exponentially ambiguous languages are $\Theta^T(2^n)$-ambiguous but not $\Theta(f)$-ambiguous for any function f. Even though each context-free grammar for an exponentially ambiguous language L is $\Theta(2^{c \cdot n})$-ambiguous for some constant $c > 0$, for a fixed constant $c > 0$ we can always find a context-free grammar which is $\Theta(2^{c' \cdot n})$-ambiguous where $c' \leq c$. Since $\Theta(2^{c_1 \cdot n}) \neq \Theta(2^{c_2 \cdot n})$ for $c_1 \neq c_2$ we cannot specify a minimum with respect to the Θ notation. It is unknown whether there are also context-free languages which are not $\Theta^T(f)$-ambiguous for any function f.

Definition 2.95 *A context-free language L is*

- unambiguous *if it is generated by an unambiguous context-free grammar.*
- finitely ambiguous, *if it is generated by a grammar with finite (degree of) ambiguity.*
- of polynomially bounded ambiguity *if it is generated by a context-free grammar with polynomially bounded ambiguity.*
- ambiguous of degree k, *if it is generated by a k-ambiguous but not by any $k-1$-ambiguous context-free grammar.*

- ambiguous, *if it is only generated by ambiguous context-free grammars.*
- infinitely ambiguous *if it is only generated by context-free grammars with infinite (degree of) ambiguity.*
- exponentially ambiguous *if it is only generated by exponentially ambiguous context-free grammars.*

The classes of unambiguous context-free languages, languages with finite ambiguity, languages with polynomially bounded ambiguity, and languages with exponential ambiguity are denoted by UCFL, FCFL, PCFL, and ECFL, respectively. As for context-free grammars we use the notions finite degree of ambiguity, infinite degree of ambiguity, *and* ambiguous of degree k *synonymously to finite ambiguity, infinite ambiguity, and k-ambiguous, respectively.*

It is well known that each context-free grammar can be generated by a grammar in Greibach normal form. Furthermore, it is obvious that each grammar in Greibach normal form is $\mathcal{O}^T(2^n)$-ambiguous. (For each derivation tree of a Greibach normal form grammar $G = (N, \Sigma, P, S)$ the number of internal nodes and leaves coincide. Thus, a word of length n has at most $|P|^n$ derivation trees.) Hence, each exponentially ambiguous language is $\Theta^T(2^n)$-ambiguous

In Section 7.4 we will see that each context-free language is either in *PCFL* or in *ECFL*. For each context-free language L and each $n \in \mathbb{N}$ it is possible to find a context-free grammar which generates L in such a way that the shortest ambiguous word is longer than n. Thus, for an ambiguous language we cannot present a least ambiguity function in the naive way. But up to a constant factor in the word length this is often possible:

Definition 2.96 *Let L be a context-free language and $f : \mathbb{N} \to \mathbb{N}$ a function. The language L is* f-ambiguous *if*

(i) there is a context-free grammar G such that $L = L(G)$ and $f = \hat{d}_G$ and

(ii) for each context-free grammar G' such that $L = L(G')$ there exists a $c \in \mathbb{N}$ such that $f(n) \leq \hat{d}_{G'}(c \cdot n)$ for all $n \in \mathbb{N} \setminus \{0\}$.

We implicitly identify the constant $k \in \mathbb{N}$ with the corresponding constant function.

Note that a language is unambiguous if it is 1-ambiguous or 0-ambiguous.

Definition 2.97 *A function $f : \mathbb{N} \to \mathbb{N}$ is an* inherent ambiguity function *if there is a context-free language L such that L is f-ambiguous.*

Note that each inherent ambiguity function is an ambiguity function and therefore it is non-decreasing. Moreover, if L is f-ambiguous then L is f'-ambiguous for each monotone function f' such that:

(i) $\{n \in \mathbb{N} \mid f(n) = 0\} = \{n \in \mathbb{N} \mid f'(n) = 0\}$

(ii) $\exists k \in \mathbb{N} : \forall n > k : f(n) = f'(n)$.

In Chapter 8 we will see that each ambiguity function of a cycle-free context-free grammar G is an inherent ambiguity function, i.e., there is a context-free language L which is $\hat{d}_G$-ambiguous. But it is not clear whether each context-free language is f-ambiguous for some function f.

2.5.4 Inherent Properties of Context-Free Languages

There is a tendency to avoid to talk about tree properties when considering context-free languages. For instance the pumping lemma is most often denoted without any reference to derivation trees, despite the fact that tree structures are crucial for its understanding. Moreover, the Lemma gets slightly weaker by eliminating the knowledge about the tree structure which is gathered in the proof. The result is an abstract lemma which is almost harder to understand than its proof. There are some good reasons to avoid to mention trees in pumping lemmata and one often heard bad reason: "By eliminating the tree property we get a pure language property." But why is a property which is shared by all the sets of derivation trees for all the context-free grammars generating a context-free language not considered as a language property? Inherent ambiguity is nothing else than an unavoidable property of the derivation tree sets of all the context-free grammars generating a fixed context-free language. Inherent ambiguity is considered as a difficult topic. The author believes that some of the difficulties are due to a sometimes unconscious refusal to accept properties of tree sets as language properties.

At first glance it is somewhat confusing to combine properties of trees and context-free languages since a language does not contain derivation trees or sets of derivation trees. On the other hand, the power set $2^{\mathbb{N}}$ does not contain a single integer, despite that nobody would deny that $2^{\mathbb{N}}$ has something to do with $\mathbb{N}$. We just have to descend one more level to see integers. It is pretty much the same for context-free languages and derivation trees:

Definition 2.98 *Let L be a context-free language. The* class of L*'s* derivation tree sets *is:*

$$\Delta_L := \{\Delta_G \mid G \text{ is a context-free grammar such that } L = L(G)\}$$

Note that Δ_L is not a set but a class since the nonterminal sets of the context-free grammars in the definition are arbitrary finite alphabets. By considering only those context-free grammars for L whose nonterminal set is a subset of a fixed infinite set we could have restricted Δ_L to a set of derivation tree sets. But there is no need to avoid a class which is not a set.

Definition 2.99 *Let L be a context-free language. A property is inherent for the language L, if it is satisfied for each element of Δ_L.*

Inherent ambiguity is just a special inherent property of a context-free language.

A well known inherent property is non-linearity of context-free languages. Each derivation tree set in Δ_L for a context-free but non-linear language has the property that it contains a tree with at least two independent internal nodes. For a context-free language which is not nonterminal bounded we can find for each $k \in \mathbb{N}$ a tree in each derivation tree set of Δ_L such that there are more than k many pairwise independent internal nodes.

2.6 Comparison with Classical Notations

In standard textbooks the ambiguity of a word is defined by its number of leftmost derivations. This is probably done to avoid the introduction of a tree formalism. It is well known that each derivation tree has a corresponding leftmost derivation. For the sake of completeness this fact is shown here for the tree formalism used in this thesis. In fact, we prove a somewhat stronger result which characterises the set of embedded trees which corresponds to a leftmost derivation. The notion of the language generated by the grammar also deviates from the classical one. But in this case the equivalence of our definition and the classical one is a direct consequence of the equivalence of the ambiguity notions for terminal strings.

For the rest of Section 2.6 we assume that $G = (N, \Sigma, P, S)$ is an arbitrary context-free grammar.

2.6.1 Classical Notation

Here we introduce definitions of standard textbooks. In case the notion is already defined in this thesis we add a prime to mark the classical formalism.

Definition 2.100 *The language generated by G is*

$$L'(G) := \left\{ w \in \Sigma^* \;\middle|\; S \Rightarrow_G^* w \right\}.$$

Definition 2.101 *For each $u \in \Sigma^*$, $\beta \in (N \cup \Sigma)^*$, and $[A, \alpha] \in P$ we define the* leftmost derivation *by:*

$$uA\beta \overset{[A,\alpha]}{\Rightarrow}_{G,lm} u\alpha\beta$$

We extend this definition of the one step derivation in the natural way to several derivation steps. Then instead of the applied productions the sequence of applied productions $\pi \in P^*$ is denoted atop the arrow Such a sequence π is called left parse.

Definition 2.102 *Let $w \in \Sigma^*$ be a terminal word. The ambiguity of w is* $d'_G(w) := |\{\pi \in P^* \mid S \overset{\pi}{\Rightarrow}_{lm} w\}|$

It is well known that each word in $L'(G)$ has at least one leftmost derivation. Hence, $L'(G) = \{w \in \Sigma^* \mid d'(w) \geq 1\}$. Similarly $L(G) = \{w \in \Sigma^* \mid d(w) \geq 1\}$. Therefore, $L(G) = L'(G)$ if $d = d'$. This equality is proved in Section 2.6.2

2.6.2 Ambiguity of Sentential Forms

First we show that the rightmost inner symbol of a tree can always be reduced:

Observation 2.103 *Let Γ be a finite alphabet and $\rho \in \Delta_\Gamma \setminus \Gamma$ a tree over Γ. Then $\rho = \tau[A, \alpha]\alpha\beta$ for some $\tau \in T^*$, $[A, \alpha] \in P_\Gamma$, and $\beta \in \Gamma^*$.*

Proof. Since $\rho \in \Delta_\Gamma \setminus \Gamma$ its derivation has at least one step. If it has exactly one step then the resulting tree has the form $[A, \alpha]\alpha$ for some $[A, \alpha] \in P_\Gamma$. This string satisfies the requirements above for $\tau = \beta = \varepsilon$. Otherwise there is a tree derivation of the form $A \overset{+}{\to} \tau'[A', \alpha']\beta' \to \rho$ where $\tau' \in T^*_\Gamma$, $[A', \alpha'] \in P_\Gamma$, and $\beta' \in \Gamma^*$. If the last expansion is applied to a symbol within the prefix τ' then $\rho = \tau[A', \alpha']\beta'$ for some $\tau \in T^*_\Gamma$, which has the required form. On the other hand if the last expansion is $\tau''A''\beta'' \to \tau''[A'', \alpha'']\alpha''\beta''$ for some $\tau'', \beta'' \in T^*_\Gamma$ and $[A'', \alpha''] \in P_\Gamma$, where $|\tau''| > |\tau'|$ then $\beta'' \in \Gamma^*$ follows such that the resulting tree has the required form in this case also. □

Now we introduce the set of trees which have a corresponding leftmost derivation.

Definition 2.104 *The set of* leftmost embedded trees *is:*

$$\mathit{leftmost}(\Delta_G) := \mathit{embedded}(\Delta_G) \cap (P \cup \Sigma)^* P (N \cup \Sigma)^*.$$

An element of $\mathit{leftmost}(\Delta_G)$ is a leftmost embedded tree.

The projection of a leftmost tree on its productions is a left parse:

Lemma 2.105 $\forall \rho \in \mathit{leftmost}(\Delta_G) : \uparrow(\rho) \overset{\pi_P(\rho)}{\Rightarrow}_{lm} \downarrow(\rho)$.

Proof. Let $\rho \in \mathit{leftmost}(\Delta_G)$. Then $|\rho|_P \geq 1$. If $|\rho|_P = 1$ then $\rho = [A, \alpha]\alpha$ for some $[A, \alpha] \in P$. Obviously, $A \overset{[A,\alpha]}{\Rightarrow}_{\mathrm{lm}} \alpha$. Moreover, $A = \uparrow(\rho)$, $\alpha = \downarrow(\rho)$, and $\pi_P(\rho) = [A, \alpha]$. Hence, $\uparrow(\rho) \overset{\pi_P(\rho)}{\Rightarrow}_{\mathrm{lm}} \downarrow(\rho)$ follows. Assume that for some integer $n \geq 1$ the statement is proved for all $\rho' \in \mathit{leftmost}(\Delta_G)$ such that $|\rho'|_P = n$. Now we consider the case $|\rho|_P = n + 1$. By Observation 2.103 we have $\rho = \tau[A, \alpha]\alpha\beta$ for some $\tau \in (P \cup N \cup \Sigma)^*$, $[A, \alpha] \in P$, and $\beta \in N \cup \Sigma^*$. Since the prefix τ of ρ is followed by a production and $\rho \in \mathit{leftmost}(\Delta_G)$ we have $\tau \in (P \cup \Sigma)^*$ which implies $\downarrow(\tau) \in \Sigma^*$. But we also have $\uparrow(\rho) \overset{*}{\rightarrow} \tau A\beta \rightarrow \tau[A, \alpha]\alpha\beta = \rho$. Now $\tau A\beta$ contains only n productions. Therefore, by the inductive hypothesis $\uparrow(\tau A\beta) \overset{\pi_P(\tau A\beta)}{\Rightarrow}_{\mathrm{lm}} \downarrow(\tau A\beta)$. Since $\uparrow(\tau A\beta) = \uparrow(\rho)$, $\pi_P(\tau A\beta) = \pi_P(\tau)$, and $\downarrow(\tau A\beta) = \downarrow(\tau)A\beta$ we can write this as $\uparrow(\rho) \overset{\pi_P(\tau)}{\Rightarrow}_{\mathrm{lm}} \downarrow(\tau)A\beta$. Moreover, $\downarrow(\tau)A\beta \Rightarrow_{\mathrm{lm}} \downarrow(\tau)\alpha\beta = \downarrow(\rho)$ since $\downarrow(\tau) \in \Sigma^*$. Hence, $\uparrow(\rho) \overset{\pi_P(\tau)[A,\alpha]}{\Rightarrow}_{\mathrm{lm}} \downarrow(\rho)$. Finally, $\pi_P(\tau)[A, \alpha] = \pi_P(\rho)$ implies $\uparrow(\rho) \overset{\pi_P(\rho)}{\Rightarrow}_{\mathrm{lm}} \downarrow(\tau)\alpha\beta = \downarrow(\rho)$ which completes the proof. □

Lemma 2.106 *The mapping π_P restricted to $\mathit{leftmost}(\Delta_G)$ is injective.*

Proof. Let $\rho_1, \rho_2 \in \mathit{leftmost}(\Delta_G)$ such that $\pi_P(\rho_1) = \pi_P(\rho_2)$. We prove by induction on the number of productions contained in ρ_1 that this equality implies $\rho_1 = \rho_2$. If $|\rho_1|_P = 1$ then $\rho_1 = \rho_2$ follows immediately. Assume the statement has been proved for $|\rho_1|_P = n \geq 1$. If $|\rho_1|_P = n + 1$ then by Observation 2.103 we have $\rho_1 = \tau[A, \alpha]\alpha\beta$ and $\rho_2 = \tau'[A', \alpha']\alpha'\beta'$ for some $[A, \alpha], [A', \alpha'] \in P$ and $\beta, \beta' \in (N \cup \Sigma)^*$. Node $|\tau| + 1$ of ρ_1 and node $|\tau'| + 1$ of ρ_2 is the last node labelled with a production respectively. Now productions are not erased by π_P. Using $\pi_P(\rho_1) = \pi_P(\rho_2)$ this leads us to $\pi_P(\tau) = \pi_P(\tau')$, $[A, \alpha] = [A', \alpha']$ and $\pi_P(\beta) = \pi_P(\beta')$. By reducing $[A, \alpha]\alpha$ to A we obtain the trees $\rho_1' := \tau A\beta$ and $\rho_2' := \tau' A\beta'$. Since $\pi_P(\rho_1') = \pi_P(\tau)A\pi(\beta) = \pi_P(\tau')A\pi(\beta') = \pi_P(\rho_2')$ and ρ_1' has one production less than ρ_1 by the inductive hypothesis we get $\rho_1' = \rho_2'$. That is $\tau A\beta = \tau' A\beta'$. If τ or τ' would contain a nonterminal then ρ_1 or ρ_2 would not be in $(P \cup \Sigma)^* P(N \cup \Sigma)^*$. This is impossible since ρ_1 and ρ_2 are leftmost embedded trees. Hence, the A to the right of τ and τ' is the leftmost nonterminal, respectively. Thus, $\tau = \tau'$ and $\beta = \beta'$. Therefore, $\rho_1 = \tau[A, \alpha]\beta = \tau'[A', \alpha']\alpha'\beta' = \rho_2$ which completes the proof. □

To show that the projection on the productions restricted to leftmost trees is surjective onto the left parses, it is sufficient to prove the following observation:

Observation 2.107 *Let $\alpha \in (N \cup \Sigma)^*$, $\pi \in P^+$, and $A \in N$. Then $A \overset{\pi}{\Rightarrow}_{lm} \alpha$ implies:*

$$\exists \rho \in \mathit{leftmost}(\Delta_G) : A \overset{*}{\to} \rho \text{ and } \downarrow(\rho) = \alpha.$$

Proof. The statement is proved by induction on the length of the left parse π. For $|\pi| = 1$ it is trivial. Assume the statement is true for $|\pi| = n \geq 1$. If $|\pi| = n + 1$ then $\pi = \pi' p$ for some $\pi' \in P^*$ and $p \in P$. Moreover, for some $\alpha' \in (N \cup \Sigma)^*$ we have $A \overset{\pi'}{\Rightarrow}_{\mathrm{lm}} \alpha' \overset{p}{\Rightarrow}_{\mathrm{lm}} \alpha$. By the inductive hypothesis there is a $\rho' \in \mathit{leftmost}(\Delta_G)$ such that $A \overset{*}{\to} \rho'$ and $\downarrow(\rho') = \alpha'$. Now ρ' is α' with some interleaved productions. Thus, we know that $\rho' \to \rho$ for some $\rho \in \mathit{embedded}(\Delta_G)$ such that $\downarrow(\rho) = \alpha$. Since the leftmost nonterminal of ρ' is replaced we even have $\rho \in \mathit{leftmost}(\Delta_G)$. Hence, we conclude with $A \overset{*}{\to} \rho \in \mathit{leftmost}(\Delta_G)$ and $\downarrow(\rho) = \alpha$. □

Finally, we have to show that $d_G = d'_G$.

Theorem 2.108 $\forall w \in \Sigma^* : d_G(w) = d'_G(w)$.

Proof. By definition $\Delta_G \subseteq \mathit{leftmost}(\Delta_G)$. Let $w \in \Sigma^*$. By Lemma 2.105 the restriction of the mapping π_P to $\{\rho \in \Delta_G \mid \downarrow(\rho) = w\}$ is a mapping into $\{\pi \in P^* \mid S \overset{*}{\Rightarrow}_{\mathrm{lm}} w\}$. By Lemma 2.106 this mapping is injective. Moreover, by Observation 2.107 it is surjective. Hence:

$$d(w) = |\{\rho \in \Delta_G \mid \downarrow(\rho) = w\}| = |\{\pi \in P^* \mid S \overset{*}{\Rightarrow}_{\mathrm{lm}} w\}| = d'_G(w).$$

□

Thus, we have shown that our definitions and the classical ones agree whenever both can be applied. In particular we have seen a one-to-one correspondence between leftmost derivations and leftmost embedded trees. However, it is possible to generalise the leftmost derivation in such a way that there is a one-to-one correspondence to all trees in $\mathit{embedded}(\Delta_G) \setminus \Sigma$. Such a generalisation can be found in [32]. For this generalised left parse the corresponding bijection is a homomorphism which erases terminals, maps nonterminals to a single "skip"-symbol s, and preserves productions. In case of a purely terminal string no s symbols occur and the generalised left parse coincides with the classical one. The use of a generalised leftmost derivation is a potential alternative to the use of a tree formalism, but experience shows that it is less convenient.

2.7 Chomsky-Schützenberger Revisited

The Chomsky-Schützenberger theorem [9] states that each context-free language L can be written as $L := h(D(\Gamma) \cap R)$ where $D(\Gamma)$ is the Dyck language over a set of parenthesis types Γ, h is a homomorphism, and R a regular set. More specific, h is a projection and $R = SF^*$ where S is a special symbol and F a finite set. The usual proof exploits the fact that each context-free language can be generated by a grammar in Greibach normal form or by the use of a Pushdown automaton. [2].

With our formalism context-free languages are defined by a projection on the set of derivation trees. Moreover, trees are closely related to parenthesis expressions. Thus, our formalisms are already close to the characterisation of Chomsky-Schützenberger. To deepen the understanding of this relationship, in the sequel an easy alternative proof of the theorem is presented. For a given context-free language L an appropriate set of the form $D(\Gamma) \cap R$ can be obtained from the set of derivation trees of *any* context-free grammar generating L essentially by the application of a homomorphism [10]. Thus, for this approach the Greibach normal form is not required and an appropriate finite set F for a language L is directly obtained from the productions of *any* context-free grammar generating L. Moreover, in contrast to the classical approach, terminals and nonterminals are treated in a uniform way which simplifies the access of the theorem. The proof deepens the understanding of both, the derivation tree formalism, and the Chomsky-Schützenberger theorem, by presenting it from a new point of view. But the results are not needed later and can therefore be omitted without loss of understanding in later chapters.

Firstly we define Dyck languages, i.e., sets of parenthesis expressions over a set of parenthesis types.

Definition 2.109 *For an arbitrary alphabet Γ we define $\bar{\Gamma} := \{\bar{a} \mid a \in \Gamma\}$ such that $|\Gamma| = |\bar{\Gamma}|$ and $\Gamma \cap \bar{\Gamma} = \emptyset$. Thus, $\bar{\Gamma}$ is a copy of the alphabet Γ. The Dyck language over Γ is $D(\Gamma) := L(G_\Gamma)$ where G_Γ is the context-free grammar*

$$G_\Gamma := S \to \bar{a}SaS \mid \varepsilon \text{ for each } a \in \Gamma.$$

A word in $D(\Gamma)$ is called a parenthesis expression *over Γ.*

Note that according to this definition symbols with a bar are left parenthesis while original symbols are right parenthesis.

[10] A leading left parenthesis for the start symbol has to be added to the homomorphic image of the derivation trees.

If $\alpha, \beta \in (\Gamma \cup \bar{\Gamma})^*$ for some alphabet Γ and $\gamma \in D(\Gamma)$, then $\alpha\gamma\beta \in D(\Gamma)$ if and only if $\alpha\beta \in D(\Gamma)$. In the sequel we exploit this well known fact implicitly.

Throughout Section 2.7 let $G := (N, \Sigma, P, S)$ be a context-free grammar and $V := N \cup \Sigma \cup \{\$\}$, where $\$$ is a symbol not in $N \cup \Sigma$. For $w \in V^*$ we define $\bar{w} := g(w)$, where $g : V^* \to \bar{V}^*$ is the homomorphism defined by $g(a) = \bar{a}$ for each $a \in V$.

Definition 2.110 *The homomorphism $h : (N \cup \Sigma \cup P)^* \to V \cup \bar{V}$ is defined by:*

$$h(X) := \begin{cases} X & \text{if } X \in N \cup \Sigma \\ \ell(X)\bar{\$}\bar{\$}\overline{r(X)}^R & \text{if } X \in P \end{cases}$$

For instance if $[A, X_1 \cdots X_k] \in P$ where $k \in \mathbb{N}$ and $\forall i \in [1, k] : X_i \in N \cup \Sigma$ then

$$h(\underbrace{[A, X_1 \cdots X_k]}_{X}) = \underbrace{A}_{\ell(X)} \bar{\$}\bar{\$} \underbrace{\bar{X}_k \cdots \bar{X}_1}_{\overline{r(X)}^R}.$$

Observation 2.111 $L(G) = \pi_\Sigma(\bar{S}h(\Delta_G))$.

Proof. Since Δ_G does not contain nonterminals we have $\downarrow(\Delta_G) = \pi_{N\cup\Sigma}(\Delta_G) = \pi_\Sigma(\Delta_G)$. Now $h(a) = a$ for each $a \in \Sigma$ and $h(p)$ does not contain a terminal symbol for any $p \in P$. Therefore, $\pi_\Sigma(h(\alpha)) = \pi_\Sigma(\alpha)$ for each $\alpha \in (P \cup \Sigma)^*$. Since $\Delta_G \subseteq (P \cup \Sigma)^*$ we finally get:

$$L(G) = \downarrow(\Delta_G) = \pi_\Sigma(\Delta_G) = \pi_\Sigma(h(\Delta_G)) = \pi_\Sigma(\bar{S}h(\Delta_G)).$$

□

To complete the proof of Chomsky Schützenberger's theorem we show that $\bar{S}h(\Delta_G)$ can be written as $D(\Gamma) \cap R$ for appropriate choices of the alphabet Γ and the regular set R.

Lemma 2.112 $\bar{S}h(\mathit{cut}(\Delta_G)) \subseteq D(V)$.

Proof. The statement is proved by the number of productions in a tree $\rho \in \mathit{cut}(\Delta_G)$. If ρ has no productions then $\rho = S$ and we have $\bar{S}h(\rho) = \bar{S}S \in D(V)$. Assume $\rho \in \mathit{cut}(\Delta_G)$ is a tree with $n \geq 1$ productions and the statement is true for all trees in $\mathit{cut}(\Delta_G)$ with less than n productions. Then by Observation 2.103 we have $\rho = \tau[A, \alpha]\alpha\beta$ for some $\tau \in T_G^*$, $\beta \in (N \cup \Sigma)^*$, and $[A, \alpha] \in P$. This implies $\rho' = \tau A\beta \in \mathit{cut}(\Delta)$. Since ρ' has one production less than ρ by the inductive hypothesis $\bar{S}h(\rho') = \bar{S}h(\tau)A\beta \in D(V)$ follows.

Finally, we prove that $\bar{S}h(\rho) \in D(V)$ by showing that $h(\rho)$ is $h(\rho')$ with a parenthesis expression inserted in front of the suffix β:

$$h(\rho) = h(\tau)h([A,\alpha])\alpha\beta = h(\tau)\underbrace{A\bar{\$}\$\bar{\alpha}^R}_{h([A,\alpha])}\alpha\beta = h(\tau)A\underbrace{\bar{\$}\$\bar{\alpha}^R\alpha}_{\in D(V)}\beta.$$

□

Lemma 2.113 *$\bar{S}h(cut(\Delta_G)) \supseteq D(V) \cap \bar{S}(N \cup \Sigma \cup h(P))^*$.*

Proof. Let $\alpha \in D(V) \cap \bar{S}(N \cup \Sigma \cup h(P))^*$. We prove $\alpha \in \bar{S}h(cut(\Delta_G))$ by induction on the number of \$ symbols in α. If α contains no \$ symbols then $\alpha \in \bar{S}(N \cup \Sigma)^*$. Thus, the leading occurrence of $\bar{S}$ is the only left parenthesis. Now $\alpha \in D(V)$ implies $\alpha = \bar{S}S = \bar{S}h(S) \in \bar{S}h(cut(\Delta_G))$. Now assume α has n occurrences of \$ symbols and the statement is true for all $\beta \in D(V) \cap \bar{S}(N \cup \Sigma \cup h(P))^*$ with less than n occurrences of \$ symbols. The last \$ symbol belongs to a $h(p)$ for some $p \in P$, i.e., $\alpha = \tau B\bar{\$}\$\bar{\beta}^R\gamma$ for some $\tau \in (V \cup \bar{V})^*$, $B \to \beta \in P$, and $\gamma \in (N \cup \Sigma)^*$. Since $\alpha \in D(V)$ this implies $\gamma = \beta\delta$ for some $\delta \in (N \cup \Sigma)^*$. Thus, $\alpha = \tau B\underbrace{\bar{\$}\$\bar{\beta}^R\beta}_{\in D(V)}\delta$ implies $\alpha' := \tau B\delta \in D(V) \cap \bar{S}(N \cup \Sigma \cup h(P))^*$. But α' has only $n-1$ many \$ symbols. Hence, $\alpha' \in \bar{S}h(cut(\Delta_G))$. Let $\rho' \in cut(\Delta_G)$ such that $\alpha' \in \bar{S}h(\rho')$. Let δ' be the shortest suffix of ρ' such that βB is a suffix of ρ'. Assume δ' contains a production. Then $\delta' = \tau'' p\delta''$ for some $\tau'' \in T_G^*$, $p \in P$ and $\delta'' \in (N \cup \Sigma)^*$. Since $h(p)$ does not contain any element of $N \cup \Sigma$ we conclude that βB is a suffix of $h(\delta'')$. But δ'' is shorter than δ' in contradiction to the choice of δ'. Hence, δ' cannot contain a production. But then $\delta' \in (N \cup \Sigma)^*$ which implies that $h(\delta') = \delta'$. Therefore, $\delta' = B\delta$. This implies that $\rho' = \tau' B\delta$ for some $\tau' \in T_G^*$ such that $h(\tau') = \tau$. Thus, $\rho := \tau'[B,\beta]\beta\delta \in cut(\Delta_G)$. Therefore, $\bar{S}h(\rho) = h(\tau')h([B,\beta])\beta\delta = \tau B\bar{\$}\$\bar{\beta}^R\gamma = \alpha$. Hence, $\alpha \in \bar{S}h(cut(\Delta_G))$. Since α was an arbitrary element of $D(V) \cap \bar{S}(N \cup \Sigma \cup h(P))^*$ the claim of the lemma holds. □

Lemma 2.114 *$\bar{S}h(\Delta_G) = D(V) \cap \bar{S}F^*$, where $F := \Sigma \cup h(P)$.*

Proof. Since $\Delta_G \subseteq cut(\Delta_G)$ we get $\bar{S}h(\Delta_G) \subseteq D(V)$ by the use of Observation 2.112. Moreover, $h(\Delta_G) \subseteq F^*$ follows immediately from $\Delta_G \subseteq (P \cup \Sigma)^*$. Thus, $\bar{S}h(\Delta_G) \subseteq D(V) \cap \bar{S}F^*$.

Now let $\gamma \in D(V) \cap \bar{S}F^*$. Then Lemma 2.113 implies that $\gamma = \bar{S}h(\rho)$ for some $\rho \in cut(\Delta_G)$. Assume $\rho \notin \Delta_G$ then $\rho = \alpha A\beta$ where $\alpha, \beta \in T_G^*$ and $A \in N$. Now $h(\rho) = h(\alpha)Ah(\beta)$. But $h(\beta) \in \{\varepsilon\} \cup (N \cup \Sigma)(V \cup \bar{V})^*$. Thus, the occurrence of A in front of the suffix $h(\beta)$ is not followed by $\bar{\$}\$$. But

$h(\rho) \in \bar{S}F^*$ and each nonterminal in a word belonging to $\bar{S}F^*$ is followed by $\bar{\$}\$$. This contradicts the assumption $\rho \notin \Delta_G$. Hence, $\rho \in \Delta_G$, which implies $\gamma \in \bar{S}h(\Delta_G)$. Since this argument applies for any $\gamma \in D(V) \cap \bar{S}F^*$ we obtain $D(V) \cap \bar{S}F^* \subseteq \bar{S}h(\Delta_G)$. □

As an immediate consequence of Lemma 2.111 and Lemma 2.114 we obtain:

Theorem 2.115 *$L(G) = \pi_\Sigma(D(V) \cap \bar{S}F^*)$, where $F := \Sigma \cup h(P)$.*

This is the theorem of Chomsky-Schützenberger. We can simplify the set F a bit by erasing the \$ parenthesis:

Definition 2.116 *Let $\Gamma := N \cup \Sigma \cup \bar{N} \cup \bar{\Sigma}$, $h' : (N \cup \Sigma \cup P)^* \to \Gamma^*$ a homomorphism defined by $h'(X) := \pi_\Gamma(h(x))$ for each $X \in N \cup \Sigma \cup P$, and $F' := \Sigma \cup h'(P)$.*

Theorem 2.117 *$L(G) = \pi_\Sigma(D(N \cup \Sigma) \cap \bar{S}F'^*)$*

Proof. Obviously, for each $\alpha \in D(V) \cap \bar{S}F^*$ we have $\pi_\Gamma(\alpha) \in D(N \cup \Sigma) \cap \bar{S}F'^*$ and $\pi_\Sigma(\alpha) = \pi_\Sigma(\pi_\Gamma(\alpha))$. Thus, $\pi_\Sigma(D(V) \cap \bar{S}F^*) \subseteq \pi_\Sigma(D(N \cup \Sigma) \cap \bar{S}F'^*)$.

On the other hand if $\beta \in D(N \cup \Sigma) \cap \bar{S}F'^*$ then by insertion of $\bar{\$}\$$ to the right of each nonterminal in β we obtain an $\alpha \in D(V) \cap \bar{S}F^*$ such that $\pi_\Gamma(\alpha) = \beta$. Again $\pi_\Sigma(\alpha) = \pi_\Sigma(\beta)$ holds, which implies $\pi_\Sigma(D(N \cup \Sigma) \cap \bar{S}F'^*) \subseteq \pi_\Sigma(D(V) \cap \bar{S}F^*)$. Thus, according to Theorem 2.115 we obtain:

$$L(G) = \pi_\Sigma(D(V) \cap \bar{S}F^*) = \pi_\Sigma(D(N \cup \Sigma) \cap \bar{S}F'^*).$$

□

Example 2.118 *Assume $G : S \to AB, \quad A \to aAb \mid \varepsilon, \quad B \to bB \mid \varepsilon$. By Theorem 2.117 we get $L(G) = \pi_\Sigma(D(N \cup \Sigma) \cap \bar{S}F'^*$, where $N := \{S, A, B\}$ and $\Sigma := \{a, b\}$. Then*

$$F' := \{a, b, \underbrace{S\bar{B}\bar{A},}_{h'([S,AB])} \underbrace{A\bar{b}\bar{A}\bar{a},}_{h'([A,aAb])} \underbrace{A,}_{h'([A,\varepsilon])} \underbrace{B\bar{B}\bar{b},}_{h'([B,bB])} \underbrace{B}_{h'([B,\varepsilon])} \}.$$

For instance we consider the word $\alpha \in D(N \cup \Sigma) \cap \bar{S}F'^*$ for which $\pi_\Sigma(\alpha) = abb$. The pairing of parenthesis is pointed out by underlines while the factorisation according to F' is reflected by the spacing:

$$\alpha = \underbrace{\underbrace{\bar{S}\quad S}\,\bar{B}\,\underbrace{\bar{A}\quad A}\,\bar{b}\,\bar{A}\,\underbrace{\bar{a}\quad a}\quad A\quad b\quad B\,\bar{B}\,\underbrace{\bar{b}\quad b}\quad B}$$

In the proof of Theorem 2.117 we have eliminated the \$ symbols used before. One can already avoid the \$ symbols in the previous lemmata and observations by using h' instead of h. But the presence of \$ symbols simplifies the proofs of Lemma 2.113 and Lemma 2.114. For ε-free grammars the proofs need only slight modifications. For grammars with ε-productions we have to take into account that h' does not map $cut(\Delta_G)$ injectively since a nonterminal A and a production $A \to \varepsilon$ have the same image under h'. This makes the argumentation somewhat more tricky and confusing. But it can be done.

The main difference between this construction and the classical one is that we keep right parenthesis of terminals instead of left ones. Now a left parenthesis $\bar{a}$ of a terminal a can be placed properly within a sequence of left parenthesis such that the corresponding terminal a occurs after saturation of the left parenthesis to the right of the corresponding $\bar{a}$ occurrence. This helps to position terminals properly. The classical approach keeps left parenthesis for terminals. Therefore, the positioning of the corresponding right parenthesis is useless and they are appended immediately to the right of the corresponding left parenthesis for the sole syntactic reason to keep the balance. Thus, the parenthesis for terminals does not contribute to the control of their positioning. This lack of control is compensated by the use of the Greibach normal form.

Chapter 3

Some Ambiguity Functions

To get a glimpse of the nature of ambiguity functions this chapter provides particular simple examples of context-free grammars with a variety of different ambiguity functions. In the second section we discuss which additional ambiguity functions can be achieved.

3.1 Right Linear Ambiguity functions

Let us start with the definition of right linear grammars and ambiguity functions:

Definition 3.1 *A context-free grammar* $G = (N, \Sigma, P, S)$ *is* right linear *if*

$$P \subseteq N \times \Sigma^*(N \cup \{\varepsilon\}).$$

An ambiguity function is right linear *if it is the ambiguity function of a right linear grammar.*

For an arbitrary integer $k > 0$ we present right linear grammars G_1, G_2, and G_3 with the following finite, polynomial, and exponential ambiguities:

grammar	$\hat{d}_G(n)$	class	$L(G)$
G_1	k	$\Theta^T(k) \subsetneq \Theta(1)$	$\{\varepsilon\}$
G_2	$\binom{n}{k}$	$\Theta(n^k) = \Theta^T(n^k)$	$a^k a^*$
G_3	k^n	$\Theta(k^n) \subsetneq \Theta^T(2^n)$	a^*

In the formulas above n represents the length of the words. The following examples are all right linear context-free grammars over a single letter alphabet.

Clearly the generated languages are regular and therefore unambiguous.[1] In this sense the ambiguity functions of the context-free grammars presented here are artificial for the corresponding languages. However, in Chapter 8 we will show that any ambiguity function of a cycle-free context-free grammar G is the inherent ambiguity function of some context-free language $L \neq L(G)$ which can be constructed from G. In this sense the trivial examples presented here, already contain essential features of inherent finite, polynomial, and exponential ambiguities.

In the sequel we represent context-free grammars by their production sets according to the convention in Section 2.3.2.

3.1.1 Finite Degree of Ambiguity

Example 3.2

$$\begin{array}{rrcl} G_1 := & S & \to & A_1 \mid \ldots \mid A_k \\ & A_1 & \to & \varepsilon \\ & & \vdots & \\ & A_k & \to & \varepsilon \end{array}$$

The grammar G_1 has start symbol S and generates only the single word ε. The set of derivation trees is $\Delta_{G_1} = \{[S, A_i][A_i, \varepsilon] \mid i \in [1, k]\}$ Thus, we have $\hat{d}_{G_1}(0) = k$. Since no other word is generated by G_1 and $\hat{d}_{G_1}(n)$ is the ambiguity of the most ambiguous word of length at most *n we have $\hat{d}_{G_1}(n) = k$ for each $n \in \mathbb{N}$.*

3.1.2 Polynomial Ambiguity

Example 3.3

$$\begin{array}{rrcl} G_2 := & A_1 & \to & aA_1 \mid aA_2 \\ & & \vdots & \\ & A_k & \to & aA_k \mid aA_{k+1} \\ & A_{k+1} & \to & aA_{k+1} \mid \varepsilon \end{array}$$

The grammar G_2 generates all the words over the singleton alphabet $\{a\}$ with at least k many a's. Each derivation starts with A_1 and each nonterminal A_i for $i \in [1, k]$ generates a sequence of a's until it switches to A_{i+1}. Finally A_{k+1} terminates the derivation after also producing a possibly void sequence of a's. Hence, a derivation tree is characterised by a vector $(n_1, \ldots, n_k, n) \in \mathbb{N}^{k+1}$, where for each $i \in [1, k]$ the integer n_i is the position of the last terminal

[1] An unambiguous grammar for a regular language R is immediately obtained from a deterministic finite automaton accepting R.

generated by a production with the nonterminal A_i on the left-hand side and n is the length of the word. For a word of length n each vector which satisfies: $1 \leq n_1 < n_2 < \cdots < n_k \leq n$ corresponds to a derivation tree. By the ordering it is sufficient to know the set of positions $\{n_i \mid i \in [1,k]\}$. Hence, the number of derivation trees for a word of length n equals the number of subsets of the interval $[1,n]$ with k elements, which is $\binom{n}{k}$. Hence, $\hat{d}_{G_2}(n) = \binom{n}{k} \in \Theta(n^k)$.

3.1.3 Exponential Ambiguity

Example 3.4

$$G_3 := \quad A_i \quad \rightarrow \quad aA_1 \mid \cdots \mid aA_k \mid \varepsilon \quad \text{for each } i \in [1,k] \quad (A_1 \text{ start symbol}).$$

The grammar G_3 generates each word in a^*. For a given $n \in \mathbb{N}$ the word a^n has the following set of derivation trees

$$\{ \quad [A_1, aX_1]a[X_1, aX_2]a \cdots [X_{n-1}, aX_n]a[X_n, \varepsilon] \mid \\ \forall i \in [1,n] : X_i \in \{A_1, \ldots, A_k\} \qquad \}.$$

Thus, we can choose for each of the n nonterminals $X_1, \ldots, X_n$ independently a nonterminal from the set $\{A_1, \ldots, A_k\}$. This yields $\hat{d}_{G_3}(n) = k^n$.

3.2 Other Ambiguity Functions

Cyclic context-free grammars can have an infinite number of derivation trees for a single word. The highest possible degree of ambiguity is obtained by the following context-free grammar:

$$G := \; S \rightarrow S \mid \varepsilon.$$

The grammar G generates the empty word, which is the shortest word at all, with infinitely many derivation trees. To the contrary, the ambiguity functions of cycle-free context-free grammars are exponentially bounded.[2] Since each context-free language can be generated by a cycle-free context-free grammar[3] super exponential ambiguity cannot be inherent for any context-free language.

[2]For cycle-free context-free grammars the number of nodes of a derivation tree is linear with respect to the length of their frontier [16, Theorem 12.2.1]. Since the trees of G are denoted by a finite alphabet the number of trees of a given length m is bounded by k^m for some $k \in \mathbb{N}$. If we sum up all the trees up to length m this geometric series is still bounded by $c \cdot k^m$ for some $c \in \mathbb{N}$. Hence, $\hat{d}_G(n) \in 2^{\mathcal{O}(n)}$.

[3]It is well known that each context-free language not containing the empty word is generated by a grammar in Greibach Normal Form, thus by a cycle-free grammar. The generation of the empty word can be added without introducing a cycle.

In Chapter 7 it will be shown that there is no ambiguity function in $2^{o(n)} \cap n^{\omega(1)}$ $(= o^T(2^n) \cap n^{\omega(1)})$.

It can be easily seen that sublinear ambiguities cannot be obtained with right linear grammars over a single letter alphabet. Thus, examples of infinite but sublinear ambiguity are necessarily more complicated than the examples presented in this chapter. However, in Chapter 6 we will see that sublinear ambiguity can be achieved with linear context-free grammars over two terminal letters. In fact, there are even rational trace languages with sublinear ambiguity.

Chapter 4

Closure Properties of Ambiguity Classes

Ambiguity classes are rather fragile with respect to natural operations. For instance the class of unambiguous context-free languages (*UCFL*) is not closed under any of the operations union, concatenation, Kleene star, length preserving homomorphisms, and projections. They are not even closed under concatenation with a two word set [16, Theorem 7.4.4]. Some rather weak closure properties can be found in [16]. For instance *UCFL* is closed under intersection with regular sets, union of disjoint sets, or concatenation and quotient with singletons. The strongest result which can be found in [16, Theorem 7.4.2] is the closure under inverse gsm mappings.

The class *FCFL* of context-free languages with finite degree of ambiguity is closed under all the operations named above. In addition, *FCFL* is obviously closed under union. But it is hard to figure out further closure properties of *FCFL*. In contrast to that *PCFL* has a certain robustness against several operations.

In this chapter we consider the ambiguity costs of several operations in more detail. As an outcome of this analysis we will find that *PCFL* is closed under union, concatenation and a restricted form of substitution, called bounded substitution. We also consider the ambiguity costs of bounded contractions which are special cases of bounded substitutions.

The costs of bounded substitutions and bounded contractions depend on the Parikh suprema of the affected symbols. Therefore, the question arises how to compute this quantity. An algorithm for the computation of Parikh suprema is presented, which is efficient with respect to the size of a context-free-grammar generating the language.

4.1 Operations with Language Constraints

This simple operation can cause such a huge ambiguity since it manipulates an unbounded number of symbols. Therefore, we define operations which are only permitted for special languages.

Definition 4.1 *Let Σ and Γ be finite alphabets. A substitution $\sigma : \Sigma^* \to 2^{\Gamma^*}$ is* bounded *for a language $L \subseteq \Sigma^*$ if for each $a \in \Sigma$ which is not bounded $\sigma(a) = \{a\}$ holds. The projection $\pi_\Gamma(L)$ is a* bounded contraction *of the language L if $\Sigma \setminus \Gamma$ contains bounded symbols only.*

Definition 4.2 *Let $L_1, L_2 \subseteq \Sigma^*$. Then $L_1 L_2$ is an* unambiguous concatenation *if for each $w \in L_1 L_2$ there is a unique $u \in L_1$ such that $w \in uL_2$*

4.2 Closure Properties of *PCFL*.

Lemma 4.3 *Let $G_1 = (N_1, \Sigma_1, P_1, S_1)$ and $G_2 = (N_2, \Sigma_2, P_2, S_2)$ be cycle-free context-free grammars. Then for $L(G_1) \circ L(G_2)$ where $\circ$ is one of the binary relations of union, unambiguous concatenation, or concatenation we can find a context-free grammar G whose ambiguity function is bounded by the expression found in the table below. Moreover, if $a \in \Sigma_1$ is bounded then for $k = \sup(L(G_1), a)$ we can find a context-free grammar G such that $L(G) = \pi_{\Sigma\setminus\{a\}} L(G_1)$ or a context-free grammar G with $L(G) = L(G_1)[a/L(G_2)]$ such that the ambiguity of G is bounded by the expression in the table below:*

operation	*ambiguity*
union	$\hat{d}_G(n) \leq \hat{d}_{G_1}(n) + \hat{d}_{G_2}(n)$
unambiguous concatenation	$\hat{d}_G(n) \leq \hat{d}_{G_1}(n)\hat{d}_{G_2}(n)$
concatenation	$\hat{d}_G(n) \leq n\hat{d}_{G_1}(n)\hat{d}_{G_2}(n)$
cancellation of a symbol	$\hat{d}_G(n) \in \mathcal{O}(n^k \hat{d}_{G_1}(n+k))$
bounded single symbol substitution	$\hat{d}_G(n) \in \mathcal{O}(n^{2k}\hat{d}_{G_1}(n+k)(\hat{d}_{G_2}(n))^k)$

Proof. Without loss of generality we can assume that $N_1 \cap N_2 = \emptyset$. Then the grammar

$$G = (N_1 \cup N_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_1 \cup P_2 \cup \{[S, S_1], [S, S_2]\}, S)$$

obviously has the property $L(G) = L(G_1) \cup L(G_2)$ and $\hat{d}_G(n) \leq \hat{d}_{G_1}(n) + \hat{d}_{G_2}(n)$ for each $n \in \mathbb{N}$. To obtain the concatenation we construct the grammar

$$G = (N_1 \cup N_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_1 \cup P_2 \cup \{[S, S_1 S_2]\}, S)$$

In general a word of length n can be split at $n+1$ many positions in two factors. Hence, we can estimate $\hat{d}_G(n)$ by a Cauchy product. For each $n \in \mathbb{N}$ we obtain:

$$\hat{d}_G(n) \leq \sum_{i=0}^{n} \hat{d}_{G_1}(i)\hat{d}_{G_2}(n-i) \leq \sum_{i=0}^{n} \hat{d}_{G_1}(n)\hat{d}_{G_2}(n) = n\hat{d}_{G_1}(n)\hat{d}_{G_2}(n).$$

In case the concatenation of $L(G_1)$ and $L(G_2)$ is unambiguous, each word has at most one valid factorisation and we obtain for each $n \in \mathbb{N}$:

$$\hat{d}_G(n) = \max\{\hat{d}_{G_1}(i)\hat{d}_{G_2}(n-i) \mid 0 \leq i \leq n\} \leq \hat{d}_{G_1}(n)\hat{d}_{G_2}(n).$$

To obtain a context-free grammar G such that $L(G) = \pi_{\Sigma\setminus\{a\}}L(G_1)$, we erase any occurrence of the symbol a from the right-hand side of a production of G_1, i.e., for $\Sigma' = \Sigma_1 \setminus \{a\}$ we choose the context-free grammar:

$$G = (N_1, \Sigma', \{[A, \pi_{\Sigma'}(\alpha)] \mid [A, \alpha] \in P\}, S_1).$$

Since a is bounded by k, a word $w \in \pi_{\Sigma'}(L(G_1)$ can be the image of words in $L(G_1)$ which have at most k many occurrences of $a's$ shuffled into w. Thus, we obtain:

$$\hat{d}_G(n) \leq \sum_{i=0}^{k} \binom{n+i}{i} \hat{d}_{G_1}(n+i) \in \mathcal{O}(n^k \hat{d}_{G_1}(n+k))$$

Finally, to obtain a context-free grammar G with $L(G) = L(G_1)[a/L(G_2)]$, we replace any occurrence of the symbol a on the right-hand side of a production of G_1 by S_2 and add all the productions and symbols of G_2. Thus, for the homomorphism $h : \Sigma_1^* \to ((\Sigma_1 \setminus \{a\}) \cup \{S_2\})^*$ defined by $h(x) = x$ for $x \in \Sigma_1 \setminus \{a\}$ and $h(a) = S_2$ we obtain the context-free grammar:

$$G = (N_1 \cup N_2, (\Sigma_1 \setminus \{a\}) \cup \Sigma_2, \{[A, h(\alpha] \mid [A, \alpha] \in P_1\} \cup P_2, S_1).$$

Now each word $w \in L(G_1)[a/L(G_2)]$ consists of a word in $L(G_1)$ where up to k many words belonging to $L(G_2)$ have been inserted. Each inserted word has a beginning and an end which can range over w. For a given factorisation of w the ambiguity which is caused by the use of the productions in the modified version of P_1 (with a replaced by S_2) is bounded by $\hat{d}_{G_1}(n+k)$, while each inserted word is produced by the productions in P_2 leading to a contribution of $\hat{d}_{G_2}(n)$ each. Thus,

$$\hat{d}_G(n) \in \mathcal{O}(n^{2k}\hat{d}_{G_1}(n+k)(\hat{d}_{G_2}(n))^k).$$

. □

Corollary 4.4 *Let $G_1 = (N_1, \Sigma_1, P_1, S_1)$ and $G_2 = (N_2, \Sigma_2, P_2, S_2)$ be cycle-free context-free grammars such that $L(G_1)L(G_2)$ is an unambiguous concatenation of $L(G_1)$ and $L(G_2)$.*

Then for $L(G_1)L(G_2)$ we can find a context-free grammar G whose ambiguity function is bounded by

$$\hat{d}_G(n) \in \Theta^T\big((\hat{d}_{G_1} \cdot \hat{d}_{G_2})(n)\big)$$

Proof. According to Lemma 4.3 we have $\hat{d}_G(n) \in \mathcal{O}^T\big((\hat{d}_{G_1} \cdot \hat{d}_{G_2})(n)\big)$. It remains to show that $\hat{d}_G(n) \in \Omega^T\big((\hat{d}_{G_1} \cdot \hat{d}_{G_2})(n)\big)$ holds. If we construct G from G_1 and G_2 as described in the proof of Lemma 4.3 we can substitute n by $2n$ in an equation proved there for unambiguous concatenations, which leads to:

$$\forall n \in \mathbb{N} : \max\{\hat{d}_{G_1}(i)\hat{d}_{G_2}(2n-i) \mid 0 \leq i \leq 2n\} = \hat{d}_G(2n).$$

In addition, we have $\hat{d}_{G_1}(n)\hat{d}_{G_2}(n) \in \{\hat{d}_{G_1}(i)\hat{d}_{G_2}(2n-i) \mid 0 \leq i \leq 2n\}$. Let $f : \mathbb{N} \to \mathbb{N}$ be defined by $f(n) = \hat{d}_G(2n)$ for each $n \in \mathbb{N}$. Then we obtain

$$f(n) = \hat{d}_G(2n) \geq \hat{d}_{G_1}(n)\hat{d}_{G_2}(n) = (\hat{d}_{G_1} \cdot \hat{d}_{G_2})(n) \in \Omega^T((\hat{d}_{G_1} \cdot \hat{d}_{G_2})(n)).$$

Since $\hat{d}_G(n) \in \Omega^T(f(n))$ holds by definition we finally obtain that $\hat{d}_G(n) \in \Omega^T(f(n)) \subseteq \Omega^T((\hat{d}_{G_1} \cdot \hat{d}_{G_2})(n))$. □

Lemma 4.5 *Let Σ and Γ be two disjoint alphabets, let $G = (N, \Sigma \cup \Gamma, P, S)$ be a context-free grammar, and let π_Σ be a bounded contraction for the language $L(G)$. Then there is a context-free grammar G' such that $L(G') = \pi_\Sigma(L)$ and $\hat{d}_{G'} \in \mathcal{O}(n^k \cdot \hat{d}_G(n+k))$, where $k := \sup(L(G))(\Gamma)$.*

Proof. Let Σ, Γ, and G be defined as mentioned above. We prove the statement by induction on $|\Gamma|$. If $|\Gamma| = 0$ then the statement is trivial. Now assume the statement is true for some $n \in \mathbb{N}$. Let $|\Gamma| = n + 1$. According to Lemma 4.3 we can cancel a symbol $a \in \Gamma$ and yield a context-free grammar G'' such that $L(G'') = \pi_{\Sigma \cup \Gamma'}(L)$ and $\hat{d}_{G''} \in \mathcal{O}(n^{k_1} \cdot \hat{d}_G(n + k_1))$, where $k_1 := \sup(L(G))(a)$ and $\Gamma' = \Gamma \setminus \{a\}$. Since $|\Gamma'| = n$ by the inductive hypothesis there is a grammar G' such that $L(G') = \pi_\Sigma(L)$ and $\hat{d}_{G'} \in \mathcal{O}(n^{k_2} \cdot \hat{d}_{G''}(n+k_2))$, where $k_2 := \sup(L(G))(\Gamma')$. Thus, for $k := k_1 + k_2 = \sup(L(G))(\Gamma)$ we obtain:

$$\hat{d}_{G'} \in \mathcal{O}\big(n^{k_2} \cdot \hat{d}_{G''}(n + k_2)\big) \subseteq \mathcal{O}\big(n^k \cdot \hat{d}_G(n + k)\big).$$

□

Corollary 4.6 *The language class PCFL is closed under bounded substitution, bounded contraction, concatenation, and union.*

Proof. Each language in *PCFL* is generated by at least one context-free grammar with a polynomially bounded ambiguity. Thus, for the union and the concatenation we can estimate $\hat{d}_{G_1}$ and $\hat{d}_{G_2}$ in Lemma 4.3 with polynomials. Which leads to a polynomial. A bounded contraction of a language $L \subseteq \Sigma^*$ which has polynomially bounded ambiguity can be simulated by at most $|\Sigma|$ many cancellations of a symbol. Hence, we obtain the product of at most $|\Sigma|$ many polynomials. Note that polynomials are homogeneous functions, which implies that for each $k \in \mathbb{N}$ and each polynomial $p(n)$ the function $q : \mathbb{N} \to \mathbb{N}$ defined by $q(n) := p(n + k)$ has the property $q(n) \in \mathcal{O}(p(n))$. A bounded substitution can also be divided in a sequence of single symbol substitutions. But we have to be cautious since a bounded substitution replaces all symbols in parallel. Thus, in contrast to a sequence of single symbol substitutions we cannot substitute a symbol which has emerged by a previous substitution. We can always simulate the parallel substitution by renaming the symbols with a sequence of single symbol substitutions such that none of the new symbols occurs in a substituted word. (Note that the renaming does not change the ambiguity.) Thus, we obtain again at most $|\Sigma|$ many products of polynomials. Again we exploit that polynomials are homogeneous. □

4.3 Semiring Closure of *UCFL*

In this section we introduce the semiring closure of unambiguous context-free languages which is the closure under union and concatenation denoted $UCFL[\cup, \cdot]$. Clearly $UCFL[\cup, \cdot]$ is a subset of *PCFL*. In this Section we will see that the inclusion is proper. This is interesting for two reasons:

- We will see in Section 5.1.5 that Earley's algorithm is capable to parse any language in the semiring closure of unambiguous context-free languages in quadratic time, i.e., for a language $L \in UCFL[\cup, \cdot]$ there is always a corresponding context-free grammar G such that $L = L(G)$ and G is parsed by the Earley parsing algorithm [11, 1] in quadratic time. If *PCFL* and $UCFL[\cup, \cdot]$ coincided no language in $PCFL$ would require more than quadratic parsing time (by the use of Earley's algorithm.) This would be a nice result.

- In Chapter 7 we will see that $PCFL$ is closed under bounded substitution. But the whole class *PCFL* can already be obtained as the closure of *UCFL* under bounded contractions. It is obvious that $UCFL[\cup, \cdot]$ is a subclass of the closure of *UCFL* under bounded contraction. But is it a strict subclass or is the closure of *UCFL* under union and concatenation already sufficient to obtain each language in *PCFL*?

Definition 4.7 *If X is a language class then $X[\cup]$ and $X[\cdot]$ denote the closure of X under union and concatenation, respectively. $X[\cdot, \cup]$ denotes the closure under union and concatenation. $X[\cdot, \cup]$ is called the semiring closure of X.*

Definition 4.8 *Let $\{a, b, c, \#\}$ be an alphabet. We define $L_\diamond := L[\$/L_p]$ where $L := \{a^n\#\$\$\#a^n \mid n \in \mathbb{N}\}$ and $L_p := \{u \in \{b, c\}^* \mid u = u^R\}$ is the language of palindromes.*

The language $L_\diamond$ already occurred as Example 2.2 to demonstrate the use of single symbol substitutions defined in Definition 2.1

Lemma 4.9 *$L_\diamond \in PCFL$.*

Proof. Let $L_1 := \{a^i\#\$\$\#a^i \mid i \in \mathbb{N}\}$. Obviously, $L_1, L_p \in UCFL$ and the substitution $[\$/L_p]$ is bounded. By Lemma 4.6 this implies $L_\diamond = L_1[\$/L_p] \in PCFL$. □

Lemma 4.10 *$L_\diamond \notin UCFL[\cup, \cdot]$.*

Proof. Assume $L_\diamond \in UCFL[\cup, \cdot]$. By the distributive laws this is equivalent to $L_\diamond \in UCFL[\cdot][\cup]$. Thus, for some $k, \ell \in \mathbb{N}$, $U_1, \ldots, U_\ell \in UCFL$, and $L_1, \ldots, L_k \in UCFL[\cdot] \setminus UCFL$ we have $L_\diamond = (\cup_{i=1}^{\ell} U_i) \cup (\cup_{i=1}^{k} L_i)$. Let us consider L_i for an arbitrary $i \in \{1, \ldots, k\}$. Now for some minimal $m \in \mathbb{N}$ we can write $L_i = \tilde{U}_1 \cdots \tilde{U}_m$ where $\tilde{U}_1, \ldots, \tilde{U}_m \in UCFL$. Since L_i is ambiguous we have $m > 1$. Each word in $L_\diamond$ contains exactly two #'s. Therefore, $\forall j \in \{1, \ldots, m\} : \forall u, v \in \tilde{U}_j : |u|_\# = |v|_\#$. Assume the words in $\tilde{U}_1$ do not contain a # then $\tilde{U}_1$ only contains words of the form a^*. Recall that for each $w \in L_i = \tilde{U}_1 \cdots \tilde{U}_m$ the number of a's to the left of the first # must match the number of a's to the right of the second #. Therefore, $\tilde{U}_1$ must be a singleton. But then $\tilde{U}_1 \cdot \tilde{U}_2$ is unambiguous contradicting the minimal choice of m. Thus, each word in $\tilde{U}_1$ must contain the first #. Similarly we obtain that each word in $\tilde{U}_m$ must contain the second #. This implies that the words in $\tilde{U}_1$ and $\tilde{U}_m$ consist of words of the forms $a^*\#\{b, c\}^*$ and $\{b, c\}^*\#a^*$, respectively. Again, if the number of a's would not be fixed we could compose words with non matching "a" blocks. Hence, $L_i \subseteq a^{n_i}\#\{b, c\}^*\#a^{n_i}$ for some $n_i \in \mathbb{N}$. We define $n = \max\{n_i \mid i \in \{1, \ldots, k\}\}+1$. Let $R := a^n\#\{b, c\}^*\#a^n$. Then $L_\diamond \cap R = ((\cup_{i=1}^{\ell} U_i) \cup (\cup_{i=1}^{k} L_i)) \cap R = ((\cup_{i=1}^{\ell} U_i) \cap R) \cup ((\cup_{i=1}^{k} L_i) \cap R)$ $= (\cup_{i=1}^{\ell} U_i) \cap R = \cup_{i=1}^{\ell} (U_i \cap R)$. Since unambiguous languages are closed under intersection with regular sets [16], this implies $L_\diamond \cap R \in UCFL[\cup]$. Moreover, unambiguous languages are closed under cancellation of singletons [16]. By cancellation of $a^n\#$ from the left-hand side and $\#a^n$ from the right-hand side, we obtain $L_pL_p \in UCFL[\cup]$. But this is false since in [10] it has been proved that L_pL_p has infinite ambiguity. Therefore, $L_\diamond \notin UCFL[\cup, \cdot]$. □

As an immediate consequence of Lemmas 4.6, 4.9 and 4.10 we obtain:

Theorem 4.11 *UCFL*$[\cup, \cdot] \subsetneq$ *PCFL.*

4.4 Computations of Parikh Suprema

Our estimation for the amount of ambiguity caused by bounded contractions and bounded substitutions in Lemma 4.3 crucially depends on the Parikh suprema of the affected symbols. If we are only interested in the question whether this amount of ambiguity is polynomially bounded or not, due to Corollary 4.6 it is sufficient just to know whether a symbol is bounded or not. How this question can be decided can be found in Section 2.3.8. But the degree of a polynomial which bounds the ambiguity of a context-free language crucially depends on Parikh suprema. Therefore, the question arises how we can determine Parikh suprema of symbols. In this section we present an efficient algorithm to compute them.

Lemma 4.12 *Let* $X \in N \cup \Sigma$. *Then* $\sup(G)(\nabla_X) = \sup(G)(X)$.

Proof. Since each element of ∇_X can generate at least one occurrence of X, we obtain $\sup(G)(\nabla_X) \leq \sup(G)(X)$. On the other hand $X \in \nabla_X$. Therefore, $\sup(G)(X) \leq \sup(G)(\nabla_X)$ is trivial. Thus, the claim follows. □

We develop a polynomial time algorithm which takes a pair, consisting of a reduced grammar $G = (N, \Sigma, P, S)$ and an alphabet $\Gamma \subseteq N \cup \Sigma$ (in a binary encoding), as the input and computes $\sup(G)(\Gamma)$. The algorithm works as follows: In the sequel we denote the pumping productions of G as $P_=$ and the descending productions as $P_{<\omega}$. First we compute *directlyPumpable* $:= \{X \in N \cup \Sigma \mid \exists p \in P_= : |r(p)|_{[X]} > |\ell(p)|_{[X]}\}$. Then we compute *Pumpable* $:= \{Y \mid X \in$ *directlyPumpable* $\wedge\, X \vdash Y\}$. It is easily seen that *Pumpable* is the set of pumpable symbols. The set $\Gamma \subseteq N \cup \Sigma$ is not bounded if and only if at least one $X \in \Gamma$ is not bounded. Hence, we can determine by the algorithm above whether Γ is bounded or not. If Γ is not bounded then $\sup(G)(\Gamma) = \omega$. Otherwise we proceed as follows: First we construct G' by erasing all productions p where $\ell(p)$ is a pumpable symbol and by erasing all occurrences of pumpable symbols in right-hand sides of productions. By Lemma 4.12 no bounded symbol can ever be generated by a pumpable symbol. Hence, G' is reduced. Again by Lemma 4.12 the bound is an invariant w.r.t. the equivalence $\equiv$. Hence, we can replace each occurrence of a symbol by its equivalence class and obtain grammar G''. The remaining pumping productions all have the form $[X] \to [X]$ and can be eliminated to obtain

G'''. It is easily seen that $\sup(G)(\Gamma) = \sup(G''')(\Gamma')$ for $\Gamma' := \{[X] \mid X \in \Gamma\}$. Since the grammar G''' has descending productions only, $\Delta(G''')$ is finite. Obviously, we can explore all the trees in $\Delta(G''')$ to compute Parikh suprema in exponential time. But we can also use memoization to compute the Parikh bound efficiently. Let $\tilde{N}$, $\tilde{\Sigma}$, $\tilde{P}$, $\tilde{S}$ be such that $G''' = (\tilde{N}, \tilde{\Sigma}, \tilde{P}, \tilde{S})$. In the sequel we allow terminals or productions as start symbols. In case of a terminal $a \in \Sigma$ the generated language is the singleton $\{a\}$, in case of a production $p \in P$ it is any word which can be derived from the right-hand side $r(p)$ of the production. Formally, we define the languages generated by grammars having terminals or productions as their start symbols as follows: For $p \in P$ we define $L((\tilde{N}, \tilde{\Sigma}, \tilde{P}, p)) := L((\tilde{N} \dot\cup \{S'\}, \tilde{\Sigma}, \tilde{P} \cup \{S' \to r(p)\}, S'))$ where $S' \notin N \cup \Sigma$ is a new nonterminal. For $a \in \Sigma$ we define $L((\tilde{N}, \tilde{\Sigma}, \tilde{P}, a)) := \{a\}$. Furthermore, we define $G_\mu := (\tilde{N}, \tilde{\Sigma}, \tilde{P}, \mu)$ for each $\mu \in \tilde{N} \cup \tilde{\Sigma} \cup \tilde{P}$. Finally, for each $\nu \in \tilde{N}$ we define $P_\nu := \{p \in \tilde{P} \mid \ell(p) = \nu\}$. We compute for each $\mu \in \tilde{P} \cup \tilde{N} \cup \tilde{\Sigma}$ the Parikh supremum of Γ' in the grammar $(\tilde{N}, \tilde{\Sigma}, \tilde{P}, \mu)$. For $\mu \in \tilde{\Sigma} \cup \Gamma'$ the task is trivial. Starting with the terminals we compute $\sup(G''')(\Gamma')$ bottom up by the use of the equation:

$$\sup(G_\mu)(\Gamma') = \begin{cases} 0 & \text{if } \mu \in \tilde{\Sigma} \setminus \Gamma' \\ 1 & \text{if } \mu \in \Gamma' \\ \max\{\sup(G_p)(\Gamma') \mid p \in P_\mu\} & \text{if } \mu \in \tilde{N} \setminus \Gamma' \\ \sum_{i=1}^{k} \sup(G_{X_i})(\Gamma') & \text{if } \mu = [A, X_1 \cdots X_k] \in P. \end{cases}$$

To talk about the complexity of the algorithm we need a reasonable measure of the size of a context-free grammar. We choose the accumulated length of right-hand sides of all productions.

We want to compute $\sup(G''')(\Gamma') = \sup(G_{\tilde{S}})(\Gamma')$. The dependency graph of G''' is a directed acyclic graph. Hence, we can compute $\sup(G_{\tilde{S}})(\Gamma')$ by a naive recursion based on the equation above. This would lead to an exponential worst case running time since many suprema would be computed several times. We can use memoization to avoid this redundancy: Let $V := \tilde{P} \cup \tilde{N} \cup \tilde{\Sigma}$. For each element of $\mu \in V$ we store in an array whether we have already computed $\sup(G_\mu)$. If this is the case we have stored the computed value in another array. At the beginning all elements of V are marked as not yet computed. We compute the supremum of elements in V by using the equation above recursively start with $\sup(G_{\tilde{S}})(\Gamma')$. Each time we need a supremum for an element of V in such a computation we first check whether we already know it. Only if the supremum is unknown we go into the recursion. If we have computed a supremum we store that information and the corresponding value. Since there are no cycles in the dependency graph of G''' we only make on recursive call for each element of V. Since all the operations of one call

can be executed in constant time, we get a linear time bound. It remains to consider the time to compute G'''. One way to do this is to compute the relation $\vdash$ first which is the transitive closure of the dependency graph. Once $\vdash$ is computed the remaining constructions can be done in linear time. Therefore, the computation of Parikh suprema is efficient.

4.5 Conclusion

As we have seen *PCFL* is closed under union, concatenation, bounded contractions and bounded substitutions. We have also seen how to compute Parikh suprema which is crucial to estimate how much ambiguity is added by the latter two operations.

Finally, with respect to the operations of regular expressions the class *PCFL* fits nicely into the hierarchy depicted below:

	$\cup$	$\cdot$	$*$
UCFL	–	–	–
FCFL	+	–	–
PCFL	+	+	–
context-free	+	+	+

Here "$\cup$" is the union, "$\cdot$" is the concatenation, and "$*$" is the Kleene-star-operation. The symbols "+" and "–" indicate whether or not a language class is closed under the corresponding operation.

Chapter 5

Ambiguity and Parsing

Parsing of context-free languages is a classical and important topic. Since this thesis is on ambiguity we will not go too deep into the theory of parsing. However, this chapter is dedicated to this subject to show some connections between parsing and ambiguity.

In Section 5.1 we revisit the well known parsing algorithm for general context-free languages due to Earley [11]. This algorithm has the nice property that it parses general context-free grammars in $\mathcal{O}(n^3)$, while reduced unambiguous context-free grammars are parsed in $\mathcal{O}(n^2)$. In both cases n is the length of the input string, while we consider the size of the grammar as a constant. Earley already showed that only a special form of ambiguity, called direct ambiguity, is expensive for the algorithm. In particular he showed that a context-free grammar with bounded direct ambiguity only requires quadratic parsing time. Note that each linear context-free grammar has bounded direct ambiguity.

We introduce an even more restricted form of ambiguity which we call immediate ambiguity. This form of ambiguity is at most linear with respect to the length of the words.[1] We will see that Earley's algorithm requires $\mathcal{O}(n^2 \cdot im_G(n))$ time to parse a context-free grammar G. Here im_G is the analogon of the ambiguity function for immediate ambiguity, i.e., for each $n \in \mathbb{N} : im_G(n)$ is the largest immediate ambiguity of a word with length at most n. We also show that $im_G(n) \leq \hat{d}_G(n + k_G)$ holds for reduced context-free grammars, where $k_G \in \mathbb{N}$ is a constant only depending on G. Thus, for reduced context-free grammars with infinite but sublinear ambiguity we obtain an Earley parsing time which is less than cubic. In Chapter 6 we will

[1] Direct ambiguity can be as large as $\mathcal{O}(n^{k-1})$, where k is the maximal number of nonterminals on the right-hand side of a production. A context-free grammar has bounded direct ambiguity if and only if it has bounded immediate ambiguity, even though the least upper bound may be lower for immediate ambiguity.

see that this is by no means a statement on an empty set. Indeed the results of Chapter 6 immediately imply the existence of an infinite hierarchy of context-free grammars with sublinear ambiguity. The ambiguity can be as low as any computable divergent total non-decreasing function. Thus, we can prove an "almost" quadratic Earley parsing time for many infinitely ambiguous context-free grammars just by considering their ambiguity function.

We can extend the upper bounds from context-free grammar classes to context-free language classes in the obvious way:

Definition 5.1 *A language class can be* Earley parsed *in time* $f : \mathbb{N} \to \mathbb{N}$ *if each language in the class is generated by at least one context-free grammar which is parsed in* $\mathcal{O}(f)$ *steps by the Earley algorithm.*

Earley showed that quadratic time is sufficient for metalinear grammars either, even though they do not have bounded direct ambiguity in general. The proof for the quadratic parsing time of metalinear languages can be immediately transferred to the closure of the class of languages with bounded direct ambiguity under union and concatenation.[2]

In Section 5.2 we use a result from [28] to show that each bounded marker language L can be parsed in logarithmic time on a CREW-PRAM with $\mathcal{O}(n^{6+k})$ many processors, where k is the so called marking constant of L. In case L has a linear witness, $\mathcal{O}(n^{2+k'})$ many processors are sufficient, where $k' \geq k$ is the so called linear marking constant for L. Obviously, the latter result is no improvement in case $k' \geq k + 4$. But often $k = k'$ holds. In Chapter 7 we will see that the class of bounded marker languages and the class of languages with polynomially bounded ambiguity ($PCFL$) coincide.

5.1 Earley Parsing Revisited

Like the Cocke-Younger-Kasami algorithm[3] (CYK-algorithm), the Earley algorithm uses dynamic programming to avoid exponential time. In the original form the CYK-algorithm parses Chomsky normal form grammars[4] in cubic time using a fixed table of quadratic size. The Earley algorithm parses any context-free grammar with at most cubic time and quadratic space. But in contrast to the CYK-algorithm the time and space requirements depend

[2] A language has bounded direct ambiguity if it is generated by a context-free grammar with bounded direct ambiguity.

[3] The first publication of the CYK-algorithm was due independently to Kasami [19] and Younger [36]. It can also be found in several textbooks, for instance [1, 17]. In these textbooks more information on Cocke's contribution is provided.

[4] A grammar $G = (N, \Sigma, P, S)$ is in Chomsky normal form if $P \subseteq N \times N^2 \cup \Sigma$.

on the parsed grammar. For instance the algorithm parses reduced linear grammars and unambiguous grammars in quadratic time without any modifications and LR(0) grammars in linear time and space. The original version [11] also allowed look ahead strings such that LR(k) grammars can be parsed in linear time and space. A good presentation for the Earley algorithm without look ahead can be found in the textbook [1, Section 4.2.2].

The basic idea can be sketched as follows: For a given context-free grammar and a given input string one can define "valid tree predicates". They represent partial tree structures which can occur in a derivation tree whose frontier matches a certain prefix of the input string. There are trivial initial valid tree predicates matching the empty prefix. Moreover, there are three rules of inference which are applicable to valid tree predicates and yield new valid tree predicates. The set of valid tree predicates turns out to be the closure of the initial tree predicates under these rules of inference. Earleys algorithm essentially computes this closure. A word is in the language if and only if the corresponding set of valid tree predicates contains tree predicates of a special form which matches the whole input string. In case that such a tree predicate exists one can compute a derivation tree in at most quadratic time from the set of valid tree predicates. (Again special cases like LR(k) grammars can be handled faster.) To see how this can be done the reader is referred to [1, Algorithm 4.6]. The proof that the set of valid tree predicates is the closure of the initial tree predicates under the three rules of inference mentioned above can be found in [1, Theorem 4.9].

In Section 5.1.1 we define valid tree predicates and the rules of inference. We do not care for the question how to compute the valid tree predicates efficiently there. For such details the reader is referred to [1, Section 4.2.2].

Section 5.1.2 provides some information concerning appropriate data structures and the handling of grammars which contain ε-productions. It is meant as a supplementation of the comments in [1]. Moreover, we will formulate the relevant aspects for the parsing by an abstraction named *duplicate*. Except for this concept the understanding of details of the implementation is not required to follow the subsequent reasoning.

In Section 5.1.3 we characterise this crucial aspect in a more mathematical way as the number of so called "completer" proofs for a valid tree predicate. By this characterisation we get rid of all the details of the implementation. Thus, the understanding of the implementation details in Section 5.1.2 is not required here. Finally, we introduce the concept of immediate ambiguity mentioned above, which is a refinement of Earley's direct ambiguity. We show that the parsing time of the Earley algorithm is bounded by $\mathcal{O}(n^2 \cdot im_G(n))$.

In Section 5.1.4 we show that immediate ambiguity is indeed a special form of ambiguity for reduced context-free grammars. For them $im_G(n) \leq$

$\hat{d}_G(n+k_G)$ holds, where $k_G \in \mathbb{N}$ is a constant only depending on G. This leads to a general upper bound of $\mathcal{O}(n^2 \cdot \hat{d}_G(n+k_G))$ for the Earley parsing time. This estimation is poor for grammars with at least linear ambiguity, since it does not improve the known cubic time bounds, but for grammars with sublinear ambiguity it is a substantial improvement, which can be obtained without analysing the rather technical immediate ambiguity.

In Section 5.1.5 we observe that Earley's proof for the quadratic parsing time of metalinear languages can be extended to the closure of languages with bounded direct ambiguity under union and concatenation.

5.1.1 Valid Tree Predicates

Definition 5.2 *Let $G = (N, \Sigma, P, S)$ be a context-free grammar. A* dotted production *is a triple $(A, \alpha, \beta) \in N \times ((N \cup \Sigma)^*)^{\times 2}$ such that $A \to \alpha\beta \in P$. We denote a dotted production (A, α, β) by $A \to \alpha \bullet \beta$. In case α or β is the empty word an explicit denotation of ε is not required, we can just omit the respective word. For instance we may write $A \to \alpha\bullet$ instead of $A \to \alpha\bullet\varepsilon$. The set of dotted productions is denoted by $P^\bullet$. Let $a_1, \ldots, a_n \in \Sigma$ for some $n \in \mathbb{N}$ and let $w := a_1 \cdots a_n$. A tree predicate is a triple $(A \to \alpha \bullet \beta, i, j) \in P^\bullet \times \mathbb{N}^2$. The set of* valid tree predicates *for the grammar G and the word w is defined by:*

$$valid_G(w) := \left\{ \begin{array}{c} (A \to \alpha \bullet \beta, i, j) \in P^\bullet \times \mathbb{N}^2 \mid \\ \exists \gamma \in (N \cup \Sigma)^* : S \overset{*}{\Rightarrow} a_1 \cdots a_i A \gamma \text{ and } \alpha \overset{*}{\Rightarrow} a_{i+1} \cdots a_j \end{array} \right\}$$

The set of initial tree predicates is:

$$initial_G := \{(S \to \bullet\beta, 0, 0) \mid \exists \beta \in (N \cup \Sigma)^* : S \to \beta \in P\}$$

It is easily seen that $initial_G \subseteq valid_G(w)$ for each $w \in \Sigma^*$. Since the number of dotted productions does not depend on the length of the words and the last two components of a valid tree predicate are non negative integers lower than $n+1$, the number of valid tree predicates is bounded by $\mathcal{O}(n^2)$. There is at most one of the three rules of inference named scanner, predictor, and completer applicable to a valid tree predicate of the form $(A, \alpha \bullet \beta, i, j)$. Which one depends on the form of β:

$\beta \in$	rule of inference
$a_{j+1}(N \cup \Sigma)^*$	scanner
$(\Sigma \setminus \{a_{j+1}\})(N \cup \Sigma)^*$	non
$N(N \cup \Sigma)^*$	predictor
ε	completer

Provided a rule of inference is applied to a valid tree predicate, the result is a valid tree predicate or a set of valid tree predicates. For arbitrary $A \in N, \alpha, \beta \in (N \cup \Sigma)^*$, and $i, j \in \mathbb{N}$ we define:

$$\begin{aligned} \text{scanner}_{G,w}(A \to \alpha \bullet a_{j+1}\beta, i, j) &:= (A \to \alpha a_{j+1} \bullet \beta, i, j+1) \\ \text{predictor}_{G,w}(A \to \alpha \bullet B\beta, i, j) &:= \{(B \to \bullet\gamma, j, j) \mid B \to \gamma \in P\} \\ \text{completer}_{G,w}(A \to \alpha\bullet, i, j) &:= \left\{ \begin{array}{c} (B \to \gamma A \bullet \delta, k, j) \mid \\ (B \to \gamma \bullet A\delta, k, i) \in \mathit{valid}_G(w) \end{array} \right\} \end{aligned}$$

The idea of the Earley algorithm is to start with $\mathit{initial}_G$ and compute the closure of this set under the three rules of inference. We denote the resulting set of tree predicates by $\mathcal{S}_{G,w}$.

For technical reasons the Earley algorithm usually does not store valid tree predicates. Instead it adds so-called states [11] or items [1] to numbered lists. A state or item is a tree predicate with the last component stripped. A state $(A \to \alpha \bullet \beta, i)$ is added to the list with number j if and only if $(A \to \alpha \bullet \beta, i, j) \in \mathcal{S}_{G,w}$.

The scanner is the only rule of inference which modifies the last component of a tree predicate. Hence, one can start by adding the initial states, which are initial tree predicates stripped off the last component, to list number 0 and complete this list by predictor and completer calls. Once a list is completed one can create a new list by scanning the items of the last completed list (,i.e., apply the scanner to them whenever possible). Then the new list is again closed under predictor and completer calls. It turns out that $\mathcal{S}_{G,w} = \mathit{valid}_G(w)$ which is shown in [11] and also in the well known textbook [1, Theorem 4.9]. By inspecting the definition, it is easily seen that $w \in L(G)$ if and only if a tree predicate of the form $(S \to \alpha\bullet, 0, n) \in \mathit{valid}_G(w)$, i.e., the item $[S \to \alpha\bullet, 0]$ is eventually added to the n-th list. This criterion can be checked easily by going through the n-th list of items once it is completed. If such an item can be found one can obtain a right parse of an appropriate derivation tree by the use of [1, Algorithm 4.6] which runs in at most quadratic time [1, Theorem 4.12].

5.1.2 Efficient Implementation

There are some pitfalls to avoid for an efficient implementation. As indicated we create the item lists one by one. The first list with number 0 is initialised by $\mathit{initial}_G$. Then we complete the 0-th list. To complete the j-th list one can go through the items of the j-th list one by one and apply an inference rule whenever possible. The new items generated are appended to the appropriate list, i.e., the j-th list in case of a predictor or completer call or the $(j+1)$-th

list in case of a scanner call. Eventually we reach the end of the j-th list. Then we go on with list $j+1$, which is already initialised by the scanner calls which have been applied to the items on the j-th list.

In case the grammar does not have ε-productions, and we are satisfied with a $\mathcal{O}(n^3)$ time complexity in any case a straightforward implementation of the algorithm described in the previous paragraph will be fine.

In case the grammar does not have ε-productions a completer call to an item of the form $(A, \alpha\bullet, i)$ on the j-th list can only occur when $i < j$. Then the i-th list is already complete and the inference rule can easily be performed by going through the i-th list. Otherwise an item $(A, \alpha\bullet, j)$ can occur on list j. The corresponding completer call should go through the j-th list which is currently under construction. Hence, some relevant items may not be created by then. One way to solve this problem is to precompute the nullable symbols, i.e. the nonterminals which can derive the empty word. Then a new rule of inference is added which applied to an item where the dot is in front of a nullable symbol creates an item where the dot is advanced by one symbol to the right. Thus, this new inference rule combines prediction and completion for nullable symbols. There are other strategies which do not involve the invention of a new rule of inference. For instance one can set a pointer at the current end of the list and store which nonterminals have been completed so far. When all the other work is done on that list we resume there and complete items which have been added since the pointer has been set.

If we want to guarantee that the predictor works in constant time we have to take into account that there may be several items on the current list with the dot in front of the same nonterminal. Therefore, if we are not careful we may create duplicate items. One can easily avoid that by an array which has a bit for each nonterminal. The values are initialised with false. A bit is set to true if a predictor call is applied to an item where the dot is immediately in front of the corresponding nonterminal. A predictor call only appends items if a prediction for the corresponding nonterminal on the current list has not occurred before.

Finally, we have to consider the completer. First of all with a similar tabular method as the predictor we should avoid the creation of duplicate items. This time we spend a bit for each possible item with the dot immediately in front of a nonterminal (Since we only have to store this information for the list under construction $\mathcal{O}(n)$ space is sufficient.) Since the number of items created is not bounded by a constant the completer cannot work in constant time. The straightforward strategy to execute the completion of an item $(A \to \alpha\bullet, i)$ on list j is to go through the i-th list and search for items where the dot is in front of the A and create an item where the dot

is advanced by one. This strategy would take $\mathcal{O}(n)$ time since the i-th list has $\mathcal{O}(i)$ entry's and $i \leq n$. Since the scanner does not require more than constant time, in any reasonable implementation, the completer is the most expensive inference rule. Since we have at most $\mathcal{O}(n^2)$ items and we do not spend more than $\mathcal{O}(n)$ for each inference rule we reach an upper bound of $\mathcal{O}(n^3)$ for the time complexity.

Now we can distribute the costs of a completer call to the items created. Sometimes this may not improve our estimation, since we may create a single item and still go through a list with $\Theta(n)$ entry's. But we can avoid to consider items where the symbol following the dot is not the required nonterminal. To achieve this each list can be divided into an array of sublists, where the symbol behind the dot (or ε in case the dot is at the right end) decides in which sublist an item is placed. Then to complete an item we can enter the sublist containing items with the dot in front of the right nonterminal. Now we can charge the costs of a completion to the generated items. But if the grammar is ambiguous we may try to create items which were already produced before. These items are charged several times with constant completion costs. Thus, the time complexity of Earley's algorithm $t_{\text{Earley}}(n) \in \mathcal{O}(n^2 \cdot (1 + \mathit{duplicate}_G(n))$, where $\mathit{duplicate}_G(n)$ is the maximum number of trials to create an item while parsing a word of length at most n.
It is easily seen that $\mathit{duplicate}_G(n) \leq 1$ for unambiguous context-free grammars thus proving an $\mathcal{O}(n^2)$ time complexity for their parsing. In Section 5.1.3 we develop upper bounds for $\mathit{duplicate}_G(n)$.

5.1.3 Immediate and Direct Ambiguity

As mentioned in Section 5.1.2 the maximum number of trials $\mathit{duplicate}(n)$ to add an item to a list is crucial for the time complexity of Earley's algorithm. We can express $\mathit{duplicate}(n)$ in a mathematically more satisfying way without considering the implementation details by the number of "completer proofs" for tree predicates. In the sequel let $G = (N, \Sigma, P, S)$ be a reduced context-free grammar, $a_1, \ldots, a_n \in \Sigma$ for some $n \in \mathbb{N}$, and $w := a_1 \cdots a_n$. Moreover, let $\vartheta := (A \to \alpha B \bullet \beta, i, j) \in \mathit{valid}_G(w)$ where $B \in N$.

Definition 5.3 *A completer proof for ϑ is a pair of valid tree predicates of the form:*

$$((B \to \gamma\bullet, k, j), (A \to \alpha \bullet B\beta, i, k))$$

We denote the number of completer proofs for ϑ by $\mathit{duplicate}_{G,w}(\vartheta)$.

Since the only variables in a completer proof for ϑ are γ and k, we easily observe that:

$$duplicate_{G,w}(\vartheta) = \left| \left\{ \begin{array}{c} (k,\gamma) \in \mathbb{N} \times (N \cup \Sigma)^* \mid \\ \alpha \stackrel{*}{\Rightarrow} a_i \cdots a_k \text{ and } \gamma \stackrel{*}{\Rightarrow} a_{k+1} \cdots a_j \text{ and } B \to \gamma \in P \end{array} \right\} \right|.$$

Since there is only a constant number of choices for γ this component is not important for the asymptotic behaviour of the time complexity. In other words, it is sufficient to consider the number of ways the generation of $a_i \cdots a_j$ is distributed between the string α and the nonterminal B. Since G is reduced each distribution is embedded in some derivation tree. To deal with this particular form of ambiguity we define:

Definition 5.4 *Let "$\bullet$" be a symbol not in $N \cup \Sigma$. Let $\alpha \in (N \cup \Sigma \cup \{\bullet\})^*$ and $w \in \Sigma^*$. Then the* marked ambiguity *of the pair (α, w) is defined by:*

$$mark_G(\alpha, w) := |\{z \in (\Sigma \cup \{\bullet\})^* \mid w = \pi_\Sigma(z) \text{ and } \alpha \stackrel{*}{\Rightarrow} z\}|$$

For $n \in \mathbb{N}$ we define $mark_G(\alpha, n) := \max\{mark_G(\alpha, w) \mid w \in \Sigma^{\leq n}\}$.

The marked ambiguity counts in how many ways the generation of a word can be shared among the strings separated by the marker symbol "$\bullet$". For $\alpha \in (N \cup \Sigma)^*$, the following relations hold:

$$mark_G(\alpha \bullet B, a_i \cdots a_j) \leq duplicate_{G,w}(\vartheta) \leq |P| \cdot mark_G(\alpha \bullet B, a_i \cdots a_j)$$
$$mark_G(\alpha \bullet B, a_i \cdots a_j) \leq mark_G(\alpha \bullet B, j - i) \leq mark_G(\alpha \bullet B, n).$$

Thus, $duplicate_{G,w}(\vartheta) \leq |P| \cdot mark_G(\alpha \bullet B, n)$ holds independently of i, j, and w. The only restriction is that $|w| \leq n$. In other words: For each $i', j' \in \mathbb{N}$ and $v \in \Sigma^{\leq n}$ we have $duplicate_{G,v}(A \to \alpha B \bullet \beta, i', j') \leq |P| \cdot mark_G(\alpha \bullet B, n)$.

Definition 5.5

$$predict\_prefix_G := \left\{ \begin{array}{c} \alpha \in (N \cup \Sigma)^* N \mid \\ \exists A \in N, \beta \in (N \cup \Sigma)^* : A \to \alpha\beta \in P \end{array} \right\}$$

$$im_G(n) := \max\{mark_G(\alpha \bullet B, n) \mid B \in N \text{ and } \alpha B \in predict\_prefix_G\}$$

Lemma 5.6

$$t_{Earley}(n) \in \mathcal{O}(n^2 \cdot (1 + im_G(n)))$$

Proof. As was said at the end of Section 5.1.2 we have

$$t_{\text{Earley}}(n) \in \mathcal{O}(n^2 \cdot (1 + duplicate_G(n)).$$

For some $B \in N$ there is a tree predicate $(A \to \alpha B \bullet \beta, i, j) \in \mathit{valid}_G$ and some $w \in \Sigma^{\leq n}$ such that the number of completer proofs is $\mathit{duplicate}_G(n)$. Then

$$\begin{array}{rcll} \mathit{duplicate}_G(n) & = & \mathit{duplicate}_{G,w}(A \to \alpha B \bullet \beta, i, j) & \\ & \leq & |P| \cdot \mathit{mark}_G(\alpha \bullet B, n) & \leq |P| \cdot \mathit{im}_G(n). \end{array}$$

Therefore, $\mathit{duplicate}_G(n) \in \mathcal{O}(\mathit{im}_G(n))$ which completes the proof □

Corollary 5.7 *A reduced linear context-free grammar $G = (N, \Sigma, P, S)$ has quadratic Earley parsing time. (Linear means that $P \subseteq N \times \Sigma^*(N \cup \{\varepsilon\})^*\Sigma^*$.)*

Proof. For a reduced linear context-free grammar $\mathrm{predict\_prefix}_G \subseteq \Sigma^* N$. But a terminal string $w \in \Sigma^*$ cannot derive anything else then w itself. Therefore, $\mathit{im}_G(n) \leq 1$ for each $n \in \mathbb{N}$. Thus, the statement follows with Lemma 5.6. □

The previous corollary is already proved by Earley in [11]. Instead of immediate ambiguity Earley introduced direct ambiguity. The notion of direct ambiguity can also be found in the more available textbook of Harrison [16], in a slightly modified way. For a context-free grammar $G = (N, \Sigma, P, S)$ Harrison defines the direct ambiguity of a pair $(p, w) \in P \times \Sigma^*$ such that it coincides with $\mathit{mark}_G(X_1 \bullet X_2 \bullet \cdots \bullet X_\ell, w)$, where $X_1, \ldots, X_\ell \in N \cup \Sigma$ and $X_1 \cdots X_\ell = r(p)$. Thus, the degree of ambiguity with respect to a production is the number of different factorisations induced by *all* the symbols on the right-hand side. A context-free grammar G has bounded degree of *direct ambiguity* if there is an upper bound which holds for any pair of the form $P \times \Sigma^*$. According to Harrison the corresponding degree is the least such upper bound. Earley defines direct ambiguity based on interfaces. The direct ambiguity of an interface $(A, w) \in N \times \Sigma^*$ coincides with the sum of direct ambiguities of the form (p, w) where p is a production with left-hand side A. The grammar G has bounded degree of direct ambiguity if there is an upper bound for all interfaces. According to Earley the corresponding degree of direct ambiguity is the least upper bound of the direct ambiguities of all interfaces. Note that we have provided two definitions for the bounded degree of direct ambiguity, once from Earley's and once from Harrison's point of view. These two definitions lead to different definitions of the finite degree of direct ambiguity, but they agree on the question whether the degree of direct ambiguity is finite or not. Hence, for the definition of the class of languages which can be generated by a context-free grammar with a bounded degree of direct ambiguity *BDCFL* the differences are irrelevant. Similarly im_G is a bounded function if and only if G has bounded degree of direct

ambiguity. In Lemma 5.6 one can replace the immediate ambiguity by direct ambiguity, and for grammars with bounded degree of direct ambiguity this is no loss of information. Things are different for unbounded immediate ambiguity. Immediate ambiguity is by definition in $\mathcal{O}(n)$. Thus, Lemma 5.6 is a refinement of the cubic time bound for the general case. In contrast to that, direct ambiguity can be as large as $\Theta(n^{j-1})$ where j is the maximum number of nonterminals on the right-hand side of a production. Therefore, a version of Lemma 5.6 with direct ambiguity is not necessarily a refinement of the cubic time result, except for grammars in Chomsky Normal Form. (The definition of Chomsky Normal Form can be found in almost in any textbook on language theory, for instance [17].) Immediate ambiguity is more suitable since the number of completer proofs only depends on the factorisation induced by the particular nonterminal over which the dot is advanced and its prefix, without any need to split this prefix any further.

5.1.4 Sublinear Ambiguity and Parsing Time

Even though immediate ambiguity is very suitable to describe the time complexity of Earley's algorithm, it is a very technical notion which is not likely to have many other applications. Can we use more natural parameters of a context-free grammar to achieve a subcubic parsing time? We already know that unambiguous context-free grammars are parsed in quadratic time while in general the time complexity is cubic. We will see in Chapter 6 that there are context-free grammars with divergent ambiguity functions in $\mathcal{O}(\log^*)$. It is natural to expect that "almost constant" functions should be parseable in "almost" quadratic time. In fact, we can show that:

Lemma 5.8 *For each reduced context-free grammar G a constant $k_G \in \mathbb{N}$ can be computed such that:*

$$\forall n \in \mathbb{N} : im_G(n) \leq \hat{d}_G(n + k_G).$$

Proof. Let $G = (N, \Sigma, P, S)$ be the considered context-free grammar and $\alpha \in \mathit{predict\_prefix}_G$. Since G is reduced $\Sigma^* \alpha \Sigma^* \cap \mathcal{S}_G \neq \emptyset$. Thus we can define the smallest number of terminal which has to be attached to the left and right of α to obtain a sentential form:

$$\begin{aligned} k_\alpha &:= \min \left\{ |\beta| - |\alpha| \,\middle|\, \beta \in \Sigma^* \alpha \Sigma^* \cap \mathcal{S}_G \right\} \\ k_G &:= \max\{k_\alpha \mid \alpha \in \mathit{predict\_prefix}_G\}. \end{aligned}$$

For each $n \in \mathbb{N}$ there are $B \in N$, $\delta \in (N \cup \Sigma)^*$ and $v \in \Sigma^{\leq n}$ such that

$$im_G(n) = mark_G(\delta \bullet B, v) \quad \text{and} \quad \delta B \in \mathit{predict\_prefix}_G.$$

Since G is reduced there is a tree $\rho \in cut(\Delta_G)$ and words $u, w \in \Sigma^*$ such that $\updownarrow_G(\rho) = [S, u\delta Bw]$ and $|uw| \leq k_G$. Thus,

$$im_G(n) \leq d_G(uvw) \leq \hat{d}_G(|uvw|) \leq \hat{d}_G(n + k_G).$$

□

As an immediate consequence of Lemma 5.6 and Lemma 5.8 we obtain:

Theorem 5.9 *For each reduced context-free grammar G there is a constant $k_G \in \mathbb{N}$ such that the parsing time of G is bounded by:*

$$t_{Earley}(n) \in \mathcal{O}(n^2 \cdot \hat{d}_G(n + k_G)).$$

Unfortunately, the constant k_G is added in Theorem 5.9 to the argument of $\hat{d}_G$ and not to the image. Therefore, we cannot replace $\hat{d}_G(n + k_G)$ by $\hat{d}_G(n)$ without any homogeneity assumptions on $\hat{d}_G$. But we talk about arbitrary ambiguity functions of cycle-free context-free grammars. Therefore, we do not know much more about $\hat{d}_G$ than the fact that they are non-decreasing and at most exponentially ambiguous. Before we define a weak restriction which is sufficient we want to define a notation:

Definition 5.10 *Let $f, g : \mathbb{N} \to \mathbb{N}$ be functions and $k_1, k_2 \in \mathbb{N}$ constants. The expressions $f(n + k_1) \in \mathcal{O}(g(n + k_2))$ and $\mathcal{O}(f(n + k_1)) \subseteq \mathcal{O}(g(n + k_2))$ mean that the functions $\tilde{f} : \mathbb{N} \to \mathbb{N}$ defined by $\tilde{f}(n) = f(n+k_1)$ and $\tilde{g} : \mathbb{N} \to \mathbb{N}$ defined by $\tilde{g}(n) = g(n+k_2)$ satisfy $\tilde{f} \in \mathcal{O}(\tilde{g})$. The free variable n denotes the parameters of the function.*

The advantage of the definition above is that it allows to handle functions for which we have not assigned a name. We could have also used the λ-calculus for this purpose. But for our rather specialised application the notation above is more intuitive and sufficient.

Definition 5.11 *A function $f : \mathbb{N} \to \mathbb{N}$ is* pointwise exponentially bounded *if $f(n + 1) \in \mathcal{O}(f(n))$.*

If f is pointwise exponentially bounded then obviously $f(n+k) \in \mathcal{O}(f(n))$ for each $k \in \mathbb{N}$. Readers familiar with semihomogenous functions, which are frequently used in the literature, may note that that each semihomogenous function is pointwise exponentially bounded, but there are pointwise exponentially bounded functions which are not semihomogenous.

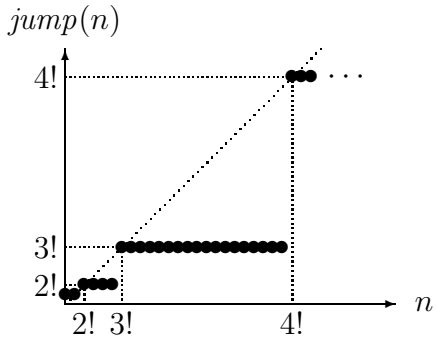


Figure 5.1: Graph of *jump*

As the name indicates super exponential functions like 2^{2^n} are not pointwise exponentially bounded. But there are also exponentially bounded functions which are not pointwise exponentially bounded. The function $jump : \mathbb{N} \to \mathbb{N}$ defined by:

$$jump(n) := (\max\{i \in \mathbb{N} \mid i! \leq n \text{ or } i = 1\})!$$

is such an example. It is depicted in Figure 5.1. One immediately observes that $jump(n) \leq n$ for each $n > 0$. Moreover, for an arbitrary divergent non-decreasing function g the function $jump \circ g \in \mathcal{O}(g)$ but $jump \circ g$ is not pointwise exponentially bounded. The function $h : \mathbb{N} \to \mathbb{N}$ defined by $h(n) := (\log^*(n))!$ is another example which is not pointwise exponentially bounded.[5]

Lemma 5.12 *If G is $\mathcal{O}(f)$-ambiguous for a pointwise exponentially bounded function $f : \mathbb{N} \to \mathbb{N}$, then $im_G(n) \in \mathcal{O}(f(n))$.*

Proof. By Lemma 5.8 we know that there is a constant k_G such that we have:

$$\forall n \in \mathbb{N} : im_G(n) \leq \hat{d}_G(n + k_G).$$

Since $\hat{d}_G \in \mathcal{O}(f)$ we obtain $\hat{d}_G(n + k_G) \in \mathcal{O}(f(n + k_G))$. Moreover, $\mathcal{O}(f(n + k_G)) \in \mathcal{O}(f)$ since f is pointwise exponentially bounded. Hence:

$$im_G(n) \in \mathcal{O}(\hat{d}_G(n + k_G)) \subseteq \mathcal{O}(f(n + k_G)) \subseteq \mathcal{O}(f(n)).$$

□

[5]The function $\log^* : \mathbb{N} \to \mathbb{N}$ is defined by $\log^*(1) := 1$ and $\log^*(n) := 1 + \log^*(\lfloor \log(n) \rfloor)$ for $n > 1$, where for each $x \in \mathbb{R}$ the value $\lfloor x \rfloor$ is the greatest integer lower then or equal to x.

Theorem 5.13 *If G is $\mathcal{O}(f)$-ambiguous for a pointwise exponentially bounded non-decreasing function $f : \mathbb{N} \to \mathbb{N}$ we obtain:*

$$t_{Earley}(n) \in \mathcal{O}(n^2 \cdot f(n)).$$

Proof. By Lemma 5.6 we have $t_{\text{Earley}}(n) \in \mathcal{O}(n^2 \cdot (1 + im_G(n)))$. By Lemma 5.12 this implies $t_{\text{Earley}}(n) \in \mathcal{O}(n^2 \cdot (1 + f(n)))$. Since G is reduced $L(G) \neq \emptyset$ implying $f(n) > 0$ for all but finitely many $n \in \mathbb{N}$. Therefore,

$$t_{\text{Earley}}(n) \in \mathcal{O}(n^2 \cdot f(n)).$$

□

Theorem 5.13 is a proper generalisation of previous known results for context-free grammars with a pointwise exponentially bounded ambiguity function $f(n) \in \omega(1) \cap o(n)$.

5.1.5 The Cost of Semiring Closures

The parsing time of Earley's algorithm consists of two parts:

(i) The number of valid tree predicates ($\mathcal{O}(n^2)$).

(ii) The number of completer proofs for a valid tree predicate ($\mathcal{O}(n)$).

To improve the time bound of Earley's algorithm for special grammar classes we have only considered the number of completer proofs so far. We have not used the fact that there are grammar classes where the total number of valid tree predicates is less than quadratic. But there are relevant grammar classes where this is the case. Earley showed for instance, that reduced $LR(0)$ grammars have at most $\mathcal{O}(n)$ many valid tree predicates. In fact, he even showed the same upper bound for $LR(k)$ grammars. But to achieve this result one has to use Earley's original version, which in contrast to the presentation in [1], exploits look ahead strings to reduce the number of completer proofs.[6] Moreover, $LR(k)$ grammars are unambiguous which implies that there is at most one completer proof for each valid tree predicate. Hence, Earley's algorithm works in $\mathcal{O}(n)$ time for each $LR(k)$ grammar if it uses a look ahead of k symbols. In particular it also works in $\mathcal{O}(n)$ time for each $LR(0)$ grammar, even in versions without look ahead.

[6]For each look ahead $k \in \mathbb{N}$ one can define a corresponding number of completer proofs for each valid tree predicate. Since the handling of look ahead strings goes beyond the scope of this thesis, we have only defined the number of completer proofs for a look ahead of 0 symbols.

Earley also showed that metalinear grammars are parsed in quadratic time. At first glance this is somewhat surprising since metalinear grammars may have $\Theta(n)$ many completer proofs for some valid tree predicates and $\Theta(n^2)$ many valid tree predicates at the same time. The trick is, that we can distribute the total costs on the dotted productions occurring in the valid tree predicate. For each dotted production with a nonterminal immediately to the left of the dot we may compute the number of corresponding items and completer proofs separately. The most expensive dotted production dominates the parsing time.

For a metalinear grammar $G = (N, \Sigma, P, S)$ we observe that the productions which have the start symbol on the left-hand side can only be used as the very first production. Therefore, the first integer component of the corresponding tree predicates is always 0. This implies that there are at most $\mathcal{O}(n)$ many such valid tree predicates. Even if they have $\Theta(n)$ many completer proofs, in total they only require $\mathcal{O}(n^2)$ time for their computation. All the other dotted productions belong to linear productions. There may be $\Theta(n^2)$ many of them but each valid tree predicate which has such a dotted production as the first component has at most one completer proof. Again this only contributes $\mathcal{O}(n^2)$ time to the computation. Hence, the total time to compute the set of valid tree predicates is $\mathcal{O}(n^2)$. The costs to retrieve a parse is also bounded by $\mathcal{O}(n^2)$. Thus, the parsing time is at most quadratic ($\mathcal{O}(n^2)$).

We generalise this technique and obtain:

Theorem 5.14 *Let $f : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function, such that $f(n) > 0$ for some $n \in \mathbb{N}$. Moreover, let $\mathcal{C}_{im}(f)$ be the class of context-free languages whose immediate ambiguity is bounded by $\mathcal{O}(f)$. Then each language in the closure of $\mathcal{C}_{im}(f)$ under union and concatenation ($\mathcal{C}_{im}(f)[\cdot, \cup]$) can be parsed with the Earley algorithm in time:*

$$t_{Earley}(n) \in \mathcal{O}(f(n) \cdot n^2).$$

Proof. Let $f : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function, such that $f(n) > 0$ for some $n \in \mathbb{N}$. Since concatenations are distributive over unions it is sufficient to consider finite unions of finite products of languages in $\mathcal{C}_{im}(f)$. Let us first consider the parsing time of finite products of languages in $\mathcal{C}_{im}(f)$, i.e., languages in the class $\mathcal{C}_{im}(f)[\cdot]$. Assume $L \in \mathcal{C}_{im}(f)[\cdot]$. Then we can write $L = L_1 \cdots L_k$ for some $k \in \mathbb{N}$ and $L_1, \ldots L_k \in \mathcal{C}_{im}(f)$. For each $i \in [1, k]$ there is a context-free grammar $G_i = (N_i, \Sigma_i, P_i, S_i)$ such that $L_i = L(G_i)$ and $im_{G_i} \in \mathcal{O}(f)$. Without loss of generality we can assume that the nonterminal sets of the grammars are pairwise disjoint. That is, for each $i, j \in [1, k]$ such that $i \neq j$ we have $N_i \cap N_j = \emptyset$. Let S be a symbol not in $\cup_{i \in [1,k]} N_i$. Now

we define $N := \cup_{i\in[1,k]} N_i$, $\Sigma := \cup_{i\in[1,k]}\Sigma_i$, and $P = \cup_{i\in[1,k]} P_i$, which we use to define the following grammar:

$$G = (N \cup \{S\},\ \Sigma,\ P \cup \{[S, S_1 \cdots S_k]\},\ S)$$

The production $[S, S_1 \cdots S_k]$ can only occur as the dominating production of a derivation tree. Hence, each valid tree predicate of the form $(S, \alpha \bullet \beta, i, j)$ has the property $i = 0$. Thus, there can be at most $\mathcal{O}(n)$ many such valid tree predicates. Moreover, each valid tree predicate has at most $\mathcal{O}(n)$ many completer proof. Thus, the production $[S, S_1 \cdots S_k]$ causes costs of at most $\mathcal{O}(n^2)$.

Let $\alpha \in (N \cup \Sigma)^*$ and $B \in N \cup \{S\}$ such that $\alpha B \in \mathit{predict\_prefix}_G$. Then $B \neq S$ since S does not occur on any right-hand side of a production of G. Moreover, there is a unique $i \in [1, k]$ such that $\alpha B \in \mathit{predict\_prefix}_{G_i}$. Furthermore, for each $n \in \mathbb{N}$ we have

$$\mathit{mark}_G(\alpha \bullet B, n) = \mathit{mark}_{G_i}(\alpha \bullet B, n) \leq \mathit{im}_{G_i}(n) \in \mathcal{O}(f)..$$

For a valid tree predicate of the form $(A, \alpha B \bullet \gamma, i, j)$ the number of completer proofs is bounded by:

$$\mathit{mark}_G(\alpha \bullet B, j) \leq \mathit{mark}_G(\alpha \bullet B, n) \in \mathcal{O}(f).$$

The number of valid tree predicates of the form $(A, \alpha B \bullet \gamma, i, j)$ is bounded by $\mathcal{O}(n^2)$. Thus, these valid tree predicates contribute $\mathcal{O}(f(n) \cdot n^2)$ time with respect to the Earley algorithm. Since no type of valid tree predicates caused more than quadratic parsing time we obtain that each language $L \in \mathcal{C}_{im}(f)[\cdot]$, which is the concatenation of finitely many languages in $\mathcal{C}_{im}(f)$, can be parsed by the Earley algorithm in time

$$t_{\text{Earley}}(n) \in \mathcal{O}(f(n) \cdot n^2).$$

If $L \in \mathcal{C}_{im}(f)[\cdot][\cup]$ then for some $m \in \mathbb{N}$ there are languages $L_1, \cdots, L_m \in \mathcal{C}_{im}(f)[\cdot]$ such that $L = \cup_{i\in[1.m]} L_i$. Thus, for each $i \in [1, m]$ there is a context-free grammar G_i, such that $L(G_i) = L_i$ and G_i can be parsed in time $\mathcal{O}(f(n) \cdot n^2)$ by the Earley algorithm. Without loss of generality we assume that the nonterminal sets of the grammars are pairwise disjoint. Now we define $N := \cup_{i\in[1,k]} N_i$, $\Sigma := \cup_{i\in[1,k]}\Sigma_i$, and $P = \cup_{i\in[1,k]} P_i$, which we use to define the following grammar:

$$G = (N \cup \{S\},\ \Sigma,\ P \cup \{[S, S_1], \ldots, [S, S_m]\},\ S)$$

The productions $[S, S_1], \ldots, [S, S_m]$ can only occure as the dominating production of a derivation tree. With a similar reasoning as above we see that

these productions does not contribute more than $\mathcal{O}(n^2)$ time to the parsing time of Earley's algorithm. With the productions with S on the left-hand side the Earley algorithm switches into a parallel parsing of the "subgrammars" G_i for $i \in [1, m]$. Thus, the resulting parsing time is:

$$\mathcal{O}(n^2) + m \cdot \mathcal{O}(f(n) \cdot n^2) = \mathcal{O}(f(n) \cdot n^2).$$

□

Corollary 5.15 *Let $f : \mathbb{N} \to \mathbb{N}$ be a non-decreasing and pointwise exponentially bounded function, such that $f(n) > 0$ for some $n \in \mathbb{N}$. Moreover, let $\mathcal{C}(f)$ be the class of context-free languages whose ambiguity is bounded by $\mathcal{O}(f)$. Then each language in the closure of $\mathcal{C}(f)$ under union and concatenation ($\mathcal{C}(f)[\cdot, \cup]$) can be parsed with the Earley algorithm in time:*

$$t_{Earley}(n) \in \mathcal{O}(f(n) \cdot n^2).$$

Proof. Using Theorem 5.14 it is sufficient to show that $\mathcal{C}(f) \subseteq \mathcal{C}_{im}(f)$. Let $L \in \mathcal{C}(f)$. Then there is a context-free grammar G which is $\mathcal{O}(f)$-ambiguous. Since f is pointwise exponentially bounded we can apply Lemma 5.12 and obtain $im_G(n) \in \mathcal{O}(f)$. Thus $L \in \mathcal{C}_{im}(f)$ proving $\mathcal{C}(f) \subseteq \mathcal{C}_{im}(f)$. □

Corollary 5.16 *The languages in the closure of BDCFL under union and concatenation can be parsed in quadratic time by the Earley algorithm, where BDCFL is the class of languages which are generated by a context-free grammar with bounded direct ambiguity.*[7]

5.2 Parallel Recognition

5.2.1 Bounded Marker Languages

Definition 5.17 *A language $L \subseteq \Sigma^*$ is a* bounded marker language *if there is an unambiguous context-free language $L' \subseteq (\Sigma \dot\cup \Gamma)^*$ such that π_Σ is a bounded contraction for L' such that $\pi_\Sigma(L') = L$. If L is a bounded marker language and L' has the properties described above then L' is a* witness *for the bounded marker language L. The* marking constant *of the witness L' for L is the maximal number of symbols which can be erased, i.e.,* $\sup(L')(\Gamma)$. *The* marking constant *of L is the least marking constant of a witness for L. The class of bounded marker languages is denoted by BMCFL.*

[7] A grammar G has bounded direct ambiguity if and only if it has bounded immediate ambiguity. i.e., there is a constant $k \in \mathbb{N}$ such that for all $n \in \mathbb{N}$ we have $im_G(n) \leq k$.

Observation 5.18 *A bounded marker language L with the marking constant k is at most $\mathcal{O}(n^k)$-ambiguous.*

Proof. Let $L \subseteq \Sigma^*$ be a bounded marker language with the marking constant k. Then there is a witness $L' \subseteq (\Sigma \dot\cup \Gamma)^*$ with a marking constant k, i.e., $k = \sup(L')(\Gamma)$. The projection π_Σ is a bounded contraction for the language L'. Since L' is a witness for L there is an unambiguous context-free grammar G such that $L(G) = L'$. Note that $\hat{d}_G(n) \leq 1$ holds for each $n \in \mathbb{N}$. According to Lemma 4.5 there is a context-free grammar G' such that $L(G') = \pi_\Sigma(L') = L$ and for $k := \sup(L(G'))(\Gamma)$ we have:

$$\hat{d}_{G'} \in \mathcal{O}(n^k \cdot \hat{d}_G(n)) \subseteq \mathcal{O}(n^k).$$

□

Observation 5.18 implies that *BMCFL* $\subseteq$ *PCFL*. At first glance *BMCFL* might appear to be an artificial subclass of *PCFL* but we will see later that *PCFL* = *BMCFL*. Thus, a context-free language has polynomially bounded ambiguity if and only if it is a bounded marker language. Clearly after the proof that *PCFL* = *BMCFL* (Theorem 7.39) the notion of bounded marker languages becomes superfluous. But to avoid forward references we do not use the equality of *PCFL* and *BMCFL* until then, which requires the temporarily use of two notions for the same thing.

5.2.2 Recognition of Bounded Marker Languages

Theorem 5.19 *The word problem of a bounded marker language L with the marking constant m_L can be solved by a CREW-PRAM with $\mathcal{O}(n^{m_L+6})$ processors in time $\mathcal{O}(\log(n))$.*[8]

Proof. Let $L \subset \Sigma^*$ be a bounded marker language with a marking constant m_L and let $L' \subseteq (\Sigma \dot\cup \Gamma)^*$ be a witness for L with marking constant m_L. For a given word $w \in \Sigma^*$ we define:

$$\mathit{insert}(w) := \{z \in (\Sigma \cup \Gamma)^* \mid w = \pi_\Sigma(z) \text{ and } |w| + m_L \geq |z|\}.$$

Thus, $\mathit{insert}(w)$ is the set of words which are obtained from inserting at most m_L many symbols from Γ into w. Then $w \in L(G)$ if and only if $\text{insert}(w) \cap L' \neq \emptyset$.

[8] CREW is short for concurrent read exclusive write. A PRAM is a parallel (P) random access machine (RAM).

In [28] Rossmanith and Rytter showed that an unambiguous context-free grammar can be recognised in time $\log(n)$ on a CREW-PRAM with $\mathcal{O}(n^6)$ processors. To determine whether a word w is in $L(G)$ we can allocate a group of $\mathcal{O}(n^6)$ processors for each element of insert(w) which are in total $\mathcal{O}(n^{m_L} \cdot n^6) = \mathcal{O}(n^{m_L+6})$ many processors. Each group of $\mathcal{O}(n^6)$ processors has the task to determine whether its corresponding element of insert(w) is in L'. Since L' is unambiguous, the algorithm of Rossmanith and Rytter can be applied for each group of $\mathcal{O}(n^6)$ processors independently. Since we have not made copies of w with inserted markers we have to explain how a processor can determine the i-th symbol of its designated word w' in constant time. (The time does not depending on the length of w but on on the constant m_L.) Each processor knows by its number how to create its designated word w' by an insertion of markers into w, i.e., it knows the at most m_L many markers and their position in w'. When a processor is asked for the i-th position of its designated word w' then there are two cases to consider:

(i) If $w'[i]$ is a marker then the processor can find out the corresponding symbol in constant time just by using its number, without access to the positions where w is stored.

(ii) If $w'[i]$ is not a marker then the processor computes in constant time the number of markers j to the left of position i and reads $w[i-j]$ concurrently.

Each group of $\mathcal{O}(n^6)$ processors examines generates a single bit answering the question whether or not its designated word belongs to L'. The computation of these bits takes logarithmic time. Now it remains to compute a logical "or" on all these bits which can be done by a tree of "or" circuits with fan-in two and $\mathcal{O}(\log(n^{m_L}) = \mathcal{O}(\log(n))$ depth. The input w is in L if and only if we get *true* as the result and the total running time is logarithmic. □

The following statement has been shown in [28, Theorem 6].

Theorem 5.20 *Each context-free language which is generated by a context-free grammar which is unambiguous and linear can be recognised in $\mathcal{O}(\log(n))$ time on a CREW-PRAM with $\mathcal{O}(n^2)$ processors.*

Unfortunately, the authors of [28] wrote down Theorem 6 in the somewhat misleading way: "Each unambiguous linear cfl can be recognized in $\mathcal{O}(\log n)$ time with n^2 processors." That they mean that this happens on a CREW-PRAM is clear from the context. It is also clear from their construction that

they need a single grammar which is at the same time linear and unambiguous. But by their statement one might be tempted to believe that it is sufficient to have a context-free language which is generated by a linear context-free grammar G_1 and by an unambiguous context-free grammar G_2. This interpretation should not be confused with the statement of Theorem 5.20. In fact, it can be shown that the language $L := \{a^iba^jba^kba^\ell \mid i = j \text{ or } k = \ell\}$ is linear context-free and unambiguous context-free, but each unambiguous context-free grammar generating it is non-linear and each linear context-free grammar generating it is ambiguous.

Theorem 5.21 *Let L be a bounded marker language L which has a witness L' having the marking constant m. In addition, we require that the witnes L' is generated by a context-free grammar which is unambiguous and linear. Then the word problem for L can be solved by a CREW-PRAM with $\mathcal{O}(n^{2+m})$ processors in time $\mathcal{O}(\log(n))$.*

Proof. Let L, L', and m satisfy the conditions required to imply the truth of the "then" portion. Then we can apply Theorem 5.20 to show that L' can be recognised in $\mathcal{O}(\log(n))$ time on a CREW-PRAM with $\mathcal{O}(n^2)$ processors. This can be used to prove the statement analogously to the proof of Theorem 5.19. □

Note that Theorem 5.21 is not necessarily an improvement compared to Theorem 5.19 even in cases where both Theorems are applicable. This is due to the fact that the constant m in Theorem 5.21 can be larger than the marking constant for the considered language L. The context-free language $L := \{a^iba^jba^kba^\ell \mid i = j \text{or } k = \ell\}$ is unambiguous and linear, but it cannot be generated by a context-free grammar which is unambiguous and linear at the same time. Hence $m_L = 0$, while 1 is a lower bound for the marking constant of an arbitrary witness of L.

Chapter 6

Sublinear Ambiguity

In this chapter it is shown that for each computable divergent total non-decreasing function $f : \mathbb{N} \to \mathbb{N}$ there is a context-free grammar G with a divergent ambiguity function g below f. This proves that extremely slowly growing ambiguity functions exist. For instance there is a context-free grammar G with infinite ambiguity below $\log^*$. In Chapter 8 we will see (Theorem 8.1) that each of these ambiguity functions is inherent for some context-free language.

6.1 The idea of the construction

It is well known that for each Turing machine M the corresponding set of valid computations can be represented as the intersection of two context-free languages. These two languages are of the Form $\mathcal{I}(\#\mathcal{R})^*$ and $(\mathcal{L}\#)^*\mathcal{F}$, respectively. Here $\mathcal{I}$ is the set of initial configurations and $\mathcal{F}$ the set of final configurations. The languages $\mathcal{L}$ and $\mathcal{R}$ generate pairs of configurations separated by a "#" symbol, such that the right configuration is obtained in one step of M from the left one. Moreover, exactly one of the configurations of such a pair is written in reverse. In case of $\mathcal{L}$ the right configuration is reversed, while for $\mathcal{R}$ it is the left one. Essentially this representation of valid computations is usually used to prove that each recursively enumerable language is the homomorphic image of the intersection of two context-free languages [3, 13, 17]. In this paper we use similar ingredients. For each Turing machine M we construct context-free grammars generating languages of the form $(\mathcal{L}\{\#\})^*\mathcal{F}(\{\#\}\mathcal{R})^*$, where $\mathcal{F}$ is an unambiguous context-free subset of M's configurations. It turns out that the ambiguity function of these grammars is dominated by the ambiguity of the words which lie in $\mathcal{F}(\#\mathcal{R})^* \cap (\mathcal{L}\#)^*\mathcal{F}$. A word w within this set represents a segment of a computation of

the Turing machine M which starts and ends in a configuration belonging to $\mathcal{F}$. Moreover, the ambiguity of w is the number of times a configuration which belongs to $\mathcal{F}$ and which is preceded by an even number of configurations occurs in w. Since we are free in the choice of the underlying Turing machine M and the unambiguous context-free set $\mathcal{F}$, we have a strong tool to design very slowly growing divergent ambiguity functions. Roughly speaking, it is sufficient to find candidates for M and $\mathcal{F}$, such that computations of M containing many configurations in $\mathcal{F}$ cannot be too short.

6.2 Preliminaries

6.2.1 Turing Machines

The reader is assumed to be familiar with single tape Turing machines as defined in [17]. For simplicity we consider deterministic machines. A *configuration* of a Turing machine consists of the tape content, the state of the Turing machine and the position of the head. It is denoted by a word consisting of the shortest string which represents the coherent portion of the tape which covers all the non blank cells and the position of the tape head. The head is denoted immediately to the left of the tape cell the machine reads in the next step. For each Turing machine M the corresponding set of configurations is denoted by ID_M. The relation $\vdash_M$ contains all the pairs of configurations $(id_a, id_b) \in ID_M \times ID_M$ where id_b is obtained from id_a by a single step of M.

6.2.2 Convention

Throughout this chapter Σ is an arbitrary finite non-empty alphabet in this chapter and the symbol $\#$ is not in Σ.

6.3 Block Correlation Languages

As a tool to design divergent ambiguity functions with a very low growth rate we introduce block correlation languages.

Definition 6.1 *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a relation. Then*

$$\mathcal{L}(R) := \{u\#v^R \mid (u,v) \in R\} \text{ and } \mathcal{R}(R) := \{u^R\#v \mid (u,v) \in R\}.^1$$

[1] Note that the letter R is used here with two meanings. As a superscript it denotes the reversal of the corresponding word. Otherwise it represents the relation R. This situation will occur frequently througout this chapter.

The relation R is (unambiguous) context-free *if $\mathcal{L}(R)$ and $\mathcal{R}(R)$ are both (unambiguous) context-free languages.*

It can be shown that $\mathcal{L}(R)$ is (unambiguous) context-free if and only if $\mathcal{R}(R)$ is (unambiguous) context-free. But instead of proving this statement for arbitrary R it is easier to check both languages for the relations considered below.

Definition 6.2 *Let $R \subseteq \Sigma^* \times \Sigma^*$ be an unambiguous context-free relation, and $\mathcal{F} \subseteq \Sigma^*$ an unambiguous context-free language. The* block correlation language *over the relation R and the set $\mathcal{F}$ is defined by:*

$$L(R, \mathcal{F}) := \big(\mathcal{L}(R)\#\big)^* \mathcal{F} \big(\#\mathcal{R}(R)\big)^*.$$

If $L(R, \mathcal{F})$ is a block-correlation language then $\mathcal{F}$ is called the corresponding language of free blocks.

Definition 6.3 *For a block correlation language $L(R, \mathcal{F})$ over a relation $R \subseteq \Sigma^* \times \Sigma^*$ and a set $\mathcal{F} \subseteq \Sigma^*$, a* canonical grammar *is a context-free grammar*

$$G := \big(\{S, A\} \,\dot\cup\, N_{\mathcal{L}} \,\dot\cup\, N_{\mathcal{R}} \,\dot\cup\, N_{\mathcal{F}},\ \Sigma \cup \{\#\},\ P \cup P_{\mathcal{L}} \cup P_{\mathcal{R}} \cup P_{\mathcal{F}},\ S\big),$$

where $G_{\mathcal{L}} := (N_{\mathcal{L}}, \Sigma \cup \{\#\}, P_{\mathcal{L}}, S_{\mathcal{L}})$, $G_{\mathcal{R}} := (N_{\mathcal{R}}, \Sigma \cup \{\#\}, P_{\mathcal{R}}, S_{\mathcal{R}})$, and $G_{\mathcal{F}} := (N_{\mathcal{F}}, \Sigma, P_{\mathcal{F}}, S_{\mathcal{F}})$ are unambiguous context-free grammars generating $\mathcal{L}(R)$, $\mathcal{R}(R)$, and $\mathcal{F}$, respectively. Moreover, P is defined by:

$$P := \{S \to S_{\mathcal{L}}\#S,\ S \to S_{\mathcal{F}}A,\ A \to A\#S_{\mathcal{R}},\ A \to \varepsilon\}.$$

Note that $G_{\mathcal{L}}$, $G_{\mathcal{R}}$, and $G_{\mathcal{F}}$ have pairwise disjoint sets of nonterminals. Moreover, these nonterminal sets do not contain the symbols S and A. This is expressed by the dot on the union symbols.

Let G be a canonical grammar which generates a block correlation language $L(R, \mathcal{F})$ with all the sets and symbols named as above and let $G' := (\{S, A\},\ \{S_{\mathcal{L}}, S_{\mathcal{R}}, S_{\mathcal{F}}, \#\},\ P, S)$. Obviously, the grammar G' is unambiguous and generates the regular language $(S_{\mathcal{L}}\#)^* S_{\mathcal{F}} (\#S_{\mathcal{R}})^*$. A given derivation tree ρ of G generating a word w can always be trimmed in a unique way to obtain a derivation tree ρ' of G'. Let α be the frontier of ρ'. If we know α we can retrieve ρ' since G' is unambiguous. Moreover, except for the first and last symbol, each occurrence of $S_{\mathcal{L}}$, $S_{\mathcal{R}}$ and $S_{\mathcal{F}}$ in α is immediately preceded and followed by a "#" symbol. Furthermore, each string of terminals generated by $S_{\mathcal{L}}$ or $S_{\mathcal{R}}$ contains exactly one "#" symbol and a terminal string generated by $S_{\mathcal{F}}$ never generates a "#" symbol. Therefore, each occurrence

of the symbols $S_{\mathcal{L}}$, $S_{\mathcal{R}}$ and $S_{\mathcal{F}}$ in α can be uniquely matched with the infix of w it generates. This is sufficient to complete the remainder of ρ uniquely since the grammars $G_{\mathcal{L}}$, $G_{\mathcal{R}}$, and $G_{\mathcal{F}}$ are unambiguous. Thus, if we know w and α we can uniquely retrieve the whole derivation tree ρ. But this does not mean that G is necessarily unambiguous since w does not determine α in general. We can only deduce the length of $\alpha \in (S_{\mathcal{L}}\#)^* S_{\mathcal{F}}(\#S_{\mathcal{R}})^*$, but there may be several permissible position for $S_{\mathcal{F}}$. We consider the "#" symbols as markers factorising w into blocks. Thus, $S_{\mathcal{F}}$ generates exactly one block which will be called the *free block* in the sequel. The free block is preceded and followed by strings of the form $\big(\mathcal{L}(R)\#\big)^*$ and $\big(\#\mathcal{R}(R)\big)^*$, respectively. (In particular this implies that the free block is preceded and followed by an even number of blocks.) The number of derivation trees for w coincides with the number of decompositions of w satisfying the requirements stated above. This number of decompositions is the canonical ambiguity of w in $L(R, F)$. More formally we define it as follows:

Definition 6.4 *Let L be a block correlation language over a relation $R \subseteq \Sigma^* \times \Sigma^*$ and a set $\mathcal{F} \subseteq \Sigma^*$. Then the* canonical ambiguity series *$d : (\Sigma \cup \{\#\})^* \to \mathbb{N}$ is defined by:*

$$d_L(w) := |\{i \in \mathbb{N} \mid w \in \big(\mathcal{L}(R)\#\big)^i \mathcal{F} \big(\#\mathcal{R}(R)\big)^*\}|.$$

With this definition we can summarise the previous considerations by the following lemma:

Lemma 6.5 *Let G be a canonical grammar generating a block correlation language L over a relation $R \subseteq \Sigma^* \times \Sigma^*$ and a set $\mathcal{F} \subseteq \Sigma^*$. Then the ambiguity function of G and the canonical ambiguity function of L are equal, i.e. $d_G = d_L$.*

Example 6.6 *Let $\Sigma = \{a\}$ and $R := \{(a^i, a^{2i}) \mid i \in \mathbb{N}\}$. Here $\mathcal{L}(R) = \mathcal{R}(R)$ since Σ is unary. To compute the ambiguity of a word w in a canonical grammar for the block correlation language $L(R, \Sigma^*)$ we consider each pair of consecutive blocks in w separated by a "#" symbol. From left to right we draw alternating arcs below and above consecutive pairs of blocks starting with an arc below the leftmost pair. An arc is drawn with a solid line if the pair is in relation, i.e., the number of a's in the right block is twice the number of a's in the left one. Otherwise the arc is dotted. Let us consider the word depicted in Figure 6.1. By definition the free block is preceded and followed by an even number of blocks. Such a block is a candidate for the free block if all the arcs below the word to its left and all the arcs above the word to its right are solid. These criteria are satisfied for exactly those blocks of w*

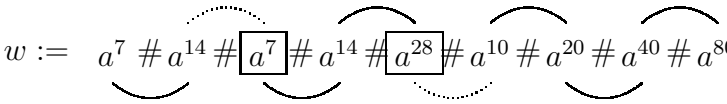


Figure 6.1: A word in the block correlation language of Example 6.6.

written in boxes. Therefore, the word w has exactly two derivation trees for any canonical grammar generating $L(R, \Sigma^*)$.

Note that there are unambiguous block correlation languages whose canonical ambiguity series is larger than 1 for some words:

Example 6.7 *Consider the unambiguous context-free relation $R := \{(a^i, a) \mid i \in \mathbb{N}\}$ over the unary alphabet $\Sigma = \{a\}$. Then it is easily seen that $L := L(R, \Sigma^*) = (a^*\#a\#)^*a^*(\#a^*\#a)^*$, which is regular, and therefore unambiguous context-free. Despite that, $d_L((a\#)^{2i}a) = i + 1$ for each $i \in \mathbb{N}$.*

Definition 6.8 *Let G be a context-free grammar over Σ. Then the* support *of $\hat{d}_G$ is the set:*

$$\mathit{support}_G := \{w \in L(G) \mid \forall u \in \Sigma^{<|w|} : d_G(u) < d_G(w)\}$$

Thus, a word w is in the support of the ambiguity function of a context-free grammar G if it is a shortest word with ambiguity at least $d_G(w)$. To determine the ambiguity function $\hat{d}_G$ of G it is sufficient to consider the words in $\mathit{support}_G$ and their corresponding ambiguities. More precisely, the ambiguity function $\hat{d}_G$ is uniquely determined by the set:

$$\{(|w|, d_G(w)) \in \mathbb{N} \times \mathbb{N} \mid w \in \mathit{support}(G)\}.$$

But how do the words in the support of a canonical grammar for a block correlation language look like? It turns out to be necessary for them that each pair of consecutive blocks is correlated. In the notation of Example 6.6 this means that a word in the support of a canonical grammar never has a "dotted" arc connecting consecutive blocks. Before showing this formally, we define:

Definition 6.9 *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a relation and $\mathcal{F} \subseteq \Sigma^*$ a formal language. Then*

$$\mathit{val}(R, \mathcal{F}) := \left\{ w_0 \# w_1^R \# \cdots \# w_{2n} \;\middle|\; \begin{array}{l} w_0, w_{2n} \in \mathcal{F} \wedge \\ \forall i \in \{0, \ldots, 2n-1\} : (w_i, w_{i+1}) \in R \end{array} \right\}.$$

It is easily seen that $val(R,\mathcal{F}) = \big(\mathcal{L}(R)\#\big)^*\mathcal{F} \cap \mathcal{F}\big(\#\mathcal{R}(R)\big)^*$ for each relation R and each language $\mathcal{F}$ over Σ.

Theorem 6.10 *Let G be a canonical grammar of some block correlation language $L(R,\mathcal{F})$. Then*

$$support_G \subseteq val(R,\mathcal{F}).$$

Proof. Let $w \in L(R,\mathcal{F})\backslash val(R,\mathcal{F})$. By definition $w \in \big(\mathcal{L}(R)\#\big)^*\mathcal{F}\big(\#\mathcal{R}(R)\big)^*$. Since $val(R,\mathcal{F}) = \big(\mathcal{L}(R)\#\big)^*\mathcal{F}\cap\mathcal{F}\big(\#\mathcal{R}(R)\big)^*$ we know that $w \notin \big(\mathcal{L}(R)\#\big)^*\mathcal{F}$ or $w \notin \mathcal{F}\big(\#\mathcal{R}(R)\big)^*$. If $w \notin \mathcal{F}\big(\#\mathcal{R}(R)\big)^*$ then $w \in \big(\mathcal{L}(R)\#\big)^+\mathcal{F}\big(\#\mathcal{R}(R)\big)^*$. But then cancellation of the first two blocks in w yields a shorter word $w' \in L(R,\mathcal{F})$. Moreover, $d_L(w) = d_L(w')$. According to Lemma 6.5 this implies $d_G(w') = d_G(w)$. Thus, w is not in the support of d_G. Analogously if $w \notin \big(\mathcal{L}(R)\#\big)^*\mathcal{F}$ we can cancel the last two blocks to obtain a shorter word w' with the same ambiguity as w which implies that w is not in the support of d_G in this case either. Thus, $(L(R,\mathcal{F}) \setminus val(R,\mathcal{F})) \cap support_G = \emptyset$. But $support_G \subseteq L(R,\mathcal{F})$. Hence, $support_G \subseteq val(R,\mathcal{F})$. □

Note that in the proof above we do neither state that the word w' obtained from the cancellation of a block pair is in $support_G$ nor that it is in $val(R,\mathcal{F})$. But since $w' \in L(R,\mathcal{F})$ we can iterate the cancellation of block pairs either from left or right until eventually a word in $val(R,\mathcal{F})$ with the same ambiguity as the original word is reached. (For the word w in Example 6.6 this procedure would yield $a^7\#a^{14}\#a^{28}$.) Since w has a finite number of blocks such an iteration terminates.

If we apply Theorem 6.10 to the language $L(R,\Sigma^*)$ of Example 6.6 we see that the support of each canonical grammar G for this language only contains words where the number of a's is doubled from block to block. That is the shortest word with ambiguity $i+1$ is $a^{2^0}\#a^{2^1}\#\cdots\#a^{2^{2i}}$ for an arbitrary $i \in \mathbb{N}$. Since the length grows exponentially with the ambiguity we see that $\hat{d}_G$ is logarithmic for each canonical context-free grammar generating $L(R,\Sigma^*)$. In fact, $L(R,\Sigma^*)$ is even inherently ambiguous of logarithmic degree. For a similar language logarithmic ambiguity has been shown in [33]. The language there is even a linear context-free language. In Section 6.6 we will introduce a block permutation *spiral* which transforms suitable block correlation languages in linear context-free languages. Up to a renaming of the symbols and a block separator symbol (here #) behind the last block the example in [33] is $spiral(L(R,\Sigma^*))$.

How can we get a divergent ambiguity function with a sublogarithmic growth rate? One trial may be to force an even stronger growth of the

length of related blocks. But this approach doesn't work as the following lemma shows:

Lemma 6.11 *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a relation and $\mathcal{L}(R)$ a context-free language with the pumping-constant n. Then*

$$\forall x, y \in \Sigma^* : \Big(\big((x,y) \in R \wedge \forall z \in \Sigma^{<|y|} : (x,z) \notin R \big) \Rightarrow n(|x|+1) \geq |y| \Big).$$

Proof. We prove this statement by induction on $|y|$. For $|y| \leq n$ it is trivial. Now assume for some $m \geq n$ the statement holds for all $y \in \Sigma^{\leq m}$. Let $y \in \Sigma^{m+1}$. For each $x \in \Sigma^*$ we have to check the implication stated above. The nontrivial case is the one where the left-hand side of the statement is satisfied. In this case $x\#y^R \in \mathcal{L}(R)$ and we can mark the rightmost $|y|$ symbols of this word. According to Ogden's Lemma [25, 17] we can pump down $x\#y^R$ into a word of the form $x'\#y' \in \mathcal{L}(R)$. Now $|y'| < |y| \leq |y'| + n$. Due to the minimality of y we obtain that $|x'| + 1 \leq |x|$. Hence, we finally get:

$$|y| = |y| - |y'| + |y'| \leq n + |y'| \leq n + n(|x'|+1) \leq n + n|x| = n(|x|+1).$$

□

Since $2|x| \geq |x| + 1$ for $x \neq \varepsilon$, Lemma 6.11 immediately implies:

$$\forall x, y \in \Sigma^+ : \Big(\big((x,y) \in R \wedge \forall z \in \Sigma^{<|y|} : (x,z) \notin R \big) \Rightarrow 2n|x| \geq |y| \Big).$$

Hence, we cannot force consecutive blocks to grow faster than by a constant factor, except for the very first step. Therefore, sublogarithmic ambiguity cannot be obtained by this method. Obviously, we can prove a version of Lemma 6.11 with $\mathcal{L}(R)$ replaced by $\mathcal{R}(R)$ in an analogous way.

6.4 Valid Computations

In Example 6.6 the language for the free blocks is Σ^*. Therefore, no candidate for the free block can be excluded in this case. As we have seen there is no hope to achieve sublogarithmic ambiguity just by increasing the growth rate of the blocks any further.

The new idea is to find an unambiguous context-free relation R and an unambiguous context-free language $\mathcal{F}$ such that in an infinite chain of words $w_0, w_1, \ldots$ such that $(w_i, w_{i+1}) \in R$ for each $i \in \mathbb{N}$ there are infinitely many words with even index belonging to the free block language $\mathcal{F}$. But with rising index the blocks in $\mathcal{F}$ occur less frequently.

Let M be a Turing machine. For each $\mathcal{F} \subseteq ID_M$ the words in $val(\vdash_M, \mathcal{F})$ represent computations which start and end in configurations belonging to $\mathcal{F}$. It is easily seen that $\vdash_M$ is an unambiguous context-free relation for each Turing machine M (even if M is nondeterministic). In fact, $\mathcal{L}(\vdash_M)$ and $\mathcal{R}(\vdash_M)$ are even deterministic and linear context-free languages. Therefore, by application of Theorem 6.10 we obtain:

Corollary 6.12 *Let M be a Turing machine and let $\mathcal{F} \subseteq ID_M$ be an unambiguous context-free language. Then $L(\vdash_M, \mathcal{F})$ is a block correlation language. Moreover, if G is a canonical grammar generating $L(\vdash_M, \mathcal{F})$ then*

$$support_G \subseteq val(\vdash_M, \mathcal{F}).$$

Even though $L(\vdash_M, \mathcal{F})$ is a large superset of $val(\vdash_M, \mathcal{F})$ we don't need to care for the words in $L(\vdash_M, \mathcal{F}) \setminus val(\vdash_M, \mathcal{F})$ since they don't contribute to the ambiguity function of G. Therefore, Corollary 6.12 provides a strong tool to design ambiguity functions. For instance let M be a Turing machine and $\mathcal{F} \subseteq ID_M$ the set of configurations of M where M is in the initial state. Let G be a canonical grammar generating $L(\vdash_M, \mathcal{F})$. Then only the words in $val(\vdash_M, \mathcal{F})$ are relevant for the computation of the ambiguity function, i.e., the words representing computations which start and end in configurations containing the initial state of M. The ambiguity of such a word is just the number of occurrences of the initial state in w at positions preceded by an even number of configurations. By the choice of $\mathcal{F}$ we can induce an additional unambiguous context-free constraint on the initial configuration.

6.5 The Design of slowly Growing Divergent Ambiguity Functions

At the end of Section 6.4 we presented an idea for the construction of a block correlation language L_M whose definition depends on an underlying Turing machine M. The ambiguity of a canonical grammar for L_M becomes small if M waits very long until it reenters the initial state. (Here waiting means that the machine enters a finite loop with the sole purpose of inducing a long computation.) Thus, the frequency of the occurrences of the initial state in a computation is roughly speaking inverse to the ambiguity of a canonical grammar for the resulting block correlation language.

Unfortunately, the inverse of a divergent non-decreasing function $f : \mathbb{N} \to \mathbb{N}$ is never a function if f is total and sublinear. Instead it is a relation. Before we develop a Turing machine construction in Section 6.5.2 in Section 6.5.1 we define the pseudo inverse of f in order to handle the inverse relationship

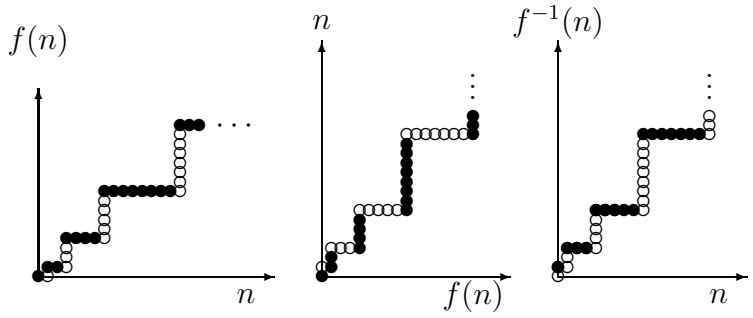


Figure 6.2: Example of a divergent non-decreasing function and its pseudo inverse.

between the initial state frequency and the resulting ambiguity. The pseudo inverse of f can be seen as the best possible divergent non-decreasing function which approximates the relation f^{-1}.

6.5.1 Pseudo Inversion of Ambiguity Functions

Definition 6.13 *Let $f : \mathbb{N} \to \mathbb{N}$ be a divergent non-decreasing function. Then the* pseudo inverse *of f is the divergent non-decreasing function $f^{-1} : \mathbb{N} \to \mathbb{N}$ defined by:*

$$f^{-1}(n) := \min\{j \in \mathbb{N} \mid n < f(j)\}$$

The definition above has a very simple graphical interpretation. In Figure 6.2 there are three graphs depicted. The filled circles of the leftmost one represent a divergent non-decreasing function f.[2] Two consecutive plateaus are connected by a chain of vertical circles which are not filled. The chain starts immediately and strictly underneath the left end of the higher plateau and ends on the level of the lower plateau. The graph in the middle is the transposition of the first one and the filled circles represent the inverse of f when we consider f as a relation. But the inverse relation is not a function. Finally, empty circles are changed into filled ones and vice versa which yields the rightmost graph. The filled circles in this graph represent the pseudo inverse f^{-1} of f according to Definition 6.13. Thus, the pseudo inverse is a divergent non-decreasing function which preserves the shape of the inverse relation of f as good as possible. A function f is the ambiguity function of a cycle-free context-free grammar G if and only if for each $n \in \mathbb{N}$ the value $f^{-1}(n)$ is the length of a shortest word whose ambiguity exceeds n.

[2] $f := \lfloor \log(n+1) \rfloor^2$

It is easy to literally *see* the truth of the following two observations by considering Figure 6.2. Formal proofs are added only for the sake of completeness and can be skipped easily without loss of understanding.

Observation 6.14 *Let $f : \mathbb{N} \to \mathbb{N}$ be a divergent non-decreasing function. Then*

$$(f^{-1})^{-1} = f.$$

Proof. Let $y, n \in \mathbb{N}$ and $S := \{z \in \mathbb{N} \mid y < f(z)\}$. First we claim that

$$n < f^{-1}(y) \Leftrightarrow f(n) \leq y.$$

The "if" and "only if" portions of this claim are proved in the next two lines:

$$\begin{array}{rl} \text{"only if:"} & n < f^{-1}(y) \;\Rightarrow\; n < \min S \;\Rightarrow\; n \notin S \;\Rightarrow\; f(n) \leq y \\ \text{"if:"} & f(n) \leq y < f(\min S) = f(f^{-1}(y)). \end{array}$$

Hence,

$$\{z \in \mathbb{N} \mid n < f^{-1}(z)\} = \{z \in \mathbb{N} \mid f(n) \leq z\}$$

and for each $n \in \mathbb{N}$ we obtain:

$$(f^{-1})^{-1}(n) = \min\{z \in \mathbb{N} \mid n < f^{-1}(z)\} = \min\{z \in \mathbb{N} \mid f(n) \leq z\} = f(n).$$

Thus, $(f^{-1})^{-1} = f$ follows. □

Observation 6.15 *Let $f, g : \mathbb{N} \to \mathbb{N}$ be two divergent non-decreasing functions. Then*

$$\forall n \in \mathbb{N} : f(n) \leq g(n) \;\Rightarrow\; g^{-1}(n) \leq f^{-1}(n)$$

Proof. Let $x, y \in \mathbb{N}$ such that $y < f(x)$ holds. Then $f(x) \leq g(x) > y$ holds, which implies:

$$\{z \in \mathbb{N} \mid y < f(z)\} \subseteq \{z \in \mathbb{N} \mid y < g(z)\}.$$

Thus, for each $y \in \mathbb{N}$ we have:

$$\begin{array}{rcl} g^{-1}(y) & = & \min\{z \in \mathbb{N} \mid y < g(z)\} \\ & \leq & \min\{z \in \mathbb{N} \mid y < f(z)\} \;=\; f^{-1}(y) \end{array}$$

□

6.5.2 Turing Machine Construction

Now we construct suitable Turing machines by the use of Corollary 6.12.

Lemma 6.16 *Let $f : \mathbb{N} \to \mathbb{N}$ be a computable divergent total non-decreasing function. Then there is a Turing machine M and an unambiguous context-free set $\mathcal{F} \subseteq ID_M$ with the following properties: For each $n \in \mathbb{N} \setminus \{0\}$ there is a word $w \in val(\vdash_M, \mathcal{F})$ which contains n occurrences of configurations in $\mathcal{F}$, and the shortest word with this property has a length of at least $f(n-1)+1$. Moreover, each occurrence of a configuration in $\mathcal{F}$ is preceded by an even number of configurations.*

Proof. Let $g : \mathbb{N} \to \mathbb{N}$ be defined by $g(n) := f(n+1)$ for each $n \in \mathbb{N}$. Then g is obviously a computable divergent total non-decreasing function. Let M' be a Turing machine which computes g. Without loss of generality we assume that $\{0,1\}$ is the input alphabet of M' and that non negative integers are encoded in binary. Let q_0 be the initial state of M. We further assume that the state sets of M and M' are disjoint. The set $\mathcal{F}$ contains all the configurations of the form $q_0 q_0' n \$ n^R$. Here \$ is a tape symbol of M which is not a tape symbol of M' and q_0' is the initial state of M' and at the same time a tape symbol of M. Finally, n is a binary encoded non negative integer. For convenience we identify binary encodings and the corresponding non-negative integers. Note that $\mathcal{F}$ is an unambiguous context-free language. The machine M never corrupts the initial format, i.e., if $\tau \in ID_M$ is reached from a configuration in $\mathcal{F}$ then by erasing the state of M from τ we obtain a string of the form $u_1 \$ u_2 \in ID_{M'} \$ \mathbb{N}$, here $\mathbb{N}$ means the set of binary encodings of non negative integers. We refer to u_1 and u_2 by calling them the *first* or *second segment* of τ. Note that u_1 contains a state of M'. The string obtained from u_1 by erasing this state is called the *tape* of the first segment. Let M be in the configuration $q_0 q_0' n_1 \$ n_1^R \in \mathcal{F}$. Then M goes through the following infinite loop:

(i) Switch to a state $q_1 \neq q_0$ which starts the simulation of M'.

(ii) Simulate M' on the first segment until it halts. This eventually happens since the function g computed by M' is total.

(iii) Wait one step and then decrement the tape of the first segment stepwise until it is 0. This is an idle loop (which loops $g(n_1)$ times when step (iii) is called for the first time. With the wait step the Turing machine requires at least $g(n_1)+1$ steps to do that.)

(iv) Increment the second segment in its reverse coding (which yields $(n_1+1)^R$ when step (iv) is called for the first time.)

(v) Overwrite the first segment by $q_0'n$, where n is the reversal of the second segment, ($q_0'n = q_0'(n_1 + 1)$ when step (v) is called for the first time), place the head at the position of q_0'. If the last time the machine M was in q_0 is an odd number of steps ago enter q_0 immediately, otherwise wait one step before entering q_0. The parity of a step can easily be stored within the finite control of the Turing machine M. This action returns M into the initial situation. Thus, it performs a kind of "**goto** (i)" command.

Moreover, we require that the Turing machine M is programmed in such a way that it does not enter the state q_0 except for the cases where this is explicitly mentioned above.

If a word $w \in \mathit{val}(\vdash_M, \mathcal{F})$ contains n occurrences of configurations in $\mathcal{F}$ for some $n \in \mathbb{N} \setminus \{0\}$ then steps (i) to (v) have been called at least $n-1$ times each. The value computed in the last call of step (ii) was $g(n_1+n-2)$, where n_1 is the argument for which g is computed the first time. Since g is non-decreasing we have $g(n_1+n-2) \geq g(n-2) = f(n-1)$, and the machine needs at least $g(n_1+n-2) \geq f(n-1)$ steps in the idle loop executed in the last call of point (iii) which is denoted in w. Hence, w contain at least $f(n-1)+1$ many configurations each of which requires at least one symbol to be denoted.

Finally, since the first configuration of a word in $\mathit{val}(\vdash_M, \mathcal{F})$ is in $\mathcal{F}$ and M always makes an even number of steps before reentering a configuration in $\mathcal{F}$, each of these configurations is preceded by an even number of configurations. This completes the proof. □

Note that M, started on a configuration in $\mathcal{F}$ runs forever and passes an infinite number of times through a configuration in $\mathcal{F}$. The set $\mathit{val}(\vdash_M, \mathcal{F})$ contains finite infixes of infinite runs of M.

The estimation in the previous proof is rather wasteful, but simple to understand. Since we are not looking for a result on the density of ambiguity functions here, we can afford to use such a rough estimation.

Theorem 6.17 *Let $f : \mathbb{N} \to \mathbb{N}$ be a computable divergent total non-decreasing function. Then there is a context-free grammar G such that $\hat{d}_G(n) \leq f^{-1}(n)$ for each $n \in \mathbb{N}$.*

Proof. By Lemma 6.16 there is a Turing machine M and an unambiguous context-free language $\mathcal{F}$ such that a shortest word in $\mathit{val}(\vdash_M, \mathcal{F})$ with n occurrences of a configuration in $\mathcal{F}$ has length at least $f(n-1)+1$ for $n \in \mathbb{N} \setminus \{0\}$. Moreover, each of these occurrences is preceded by an even number of configurations. Therefore, according to Lemma 6.5 these words

have n derivation trees in a canonical grammar G generating $L(\vdash_M, \mathcal{F})$ and by Corollary 6.12 we do not need to consider other words in $L(\vdash_M, \mathcal{F})$. In other words the shortest words generated by G whose ambiguity exceeds n have length at least $f(n)+1$. Thus $\hat{d}_G(n) \leq f^{-1}(n)$ for each $n \in \mathbb{N}$. □

Theorem 6.18 *If f is a computable divergent total non-decreasing function then there is a cycle-free context-free grammar G such that $\hat{d}_G$ is a divergent function satisfying $\hat{d}_G(n) \leq f(n)$ for all $n \in \mathbb{N}$.*

Proof. Let f be a computable divergent total non-decreasing function. Let $g := f^{-1}$. Obviously g is also a computable divergent total non-decreasing function. According to Theorem 6.17 and Observation 6.14 there is a context-free grammar G such that $\hat{d}_G(n) \leq g^{-1}(n) = ((f^{-1})^{-1})(n) = f(n)$ for each $n \in \mathbb{N}$. □

6.6 Linearisation

We can obtain the result of Theorem 6.18 even with a more restricted type of grammars. More specific instead of cycle-free context-free grammars we can use linear cycle-free context-free grammars with a so-called unambiguous turn position. In order to understand properly what that means and how this result is achieved we need a few definitions.

6.6.1 Linear Grammar with Unambiguous Turn Position

Definition 6.19 *A linear context-free grammar $G = (N, \Sigma, P, S)$ has* erasing termination *if*

$$P \subseteq N \times (\Sigma^* N \Sigma^* \cup \{\varepsilon\})$$

Obviously, each linear context-free grammar can be transformed into an equivalent linear context-free grammar which has erasing termination. One just needs to add a new nonterminal A to the right-hand side of each inappropriate production and add the production $A \to \varepsilon$.

Definition 6.20 *Let $G = (N, \Sigma, P, S)$ be a context-free grammar with erasing termination and $w \in L(G)$. Then $n \in \mathbb{N}$ is a* turn position *of w according to G if there are $w_1 \in \Sigma^n$ and $w_2 \in \Sigma^*$ such that $S \overset{*}{\Rightarrow} w_1 A w_2 \Rightarrow w$ for some $A \in N$. Obviously, this implies $w = w_1 w_2$. Therefore, we call w_1 the* left portion *of the turn n of w and w_2 the* right portion *of the turn n of w.*

Definition 6.21 *A linear context-free grammar $G = (N, \Sigma, P, S)$ with erasing termination has an* unambiguous turn position *if each word $w \in L(G)$ has a unique turn position according to G. Let $w \in L(G)$ where G is a linear context-free grammar with unambiguous turn position. Then the* left *and* right portion *of w according to G is the left and right portion of the unique turn of w according to G, respectively.*

The name "turn" is motivated by one-turn push down automata, which are a machine model for linear context-free languages. One-turn pushdown automata are usual pushdown automata except for the fact that they allow only one turn from pushing to popping, i.e. they work in two phases. In the first phase the content of the pushdown does not shrink, in the second phase the content does not grow. A linear context-free grammar with erasing termination can easily be transformed into a one-turn pushdown automaton such that the left portion of a turn position of a word w is the input read during the pushing phase of some successful computation τ of w, while the right portion of the turn position is read during the popping phase of the computation τ. One-turn push down automata are a special case of finite-turn push down automate which allow a finite number of turns from pushing to popping and vice versa. This type of restriction for the way to work with a pushdown is also called "reversal bounded" in the literature [3]. Finite-turn pushdown automata are the machine model corresponding to nonterminal bounded context-free languages [16, Section 5.7, Exercise 7 page 212].

Definition 6.22 *A linear context-free language has a* fixable turn position *if it is generated by some linear context-free grammar with unambiguous turn position.*

An example for a language which does not have a fixable turn position is $\{a^i b^j c^k \mid i = j \text{ or } j = k\}$ while $\{a^i b^j c^k \mid i = j \text{ or } i = k\}$ has a fixable turn position.

6.6.2 Spiral Permutation

As we have seen in Example 6.7 there are special cases of block correlation languages which are regular. But typically a block correlation language $L(R, \mathcal{F})$ over $R \subseteq \Sigma^* \times \Sigma^*$ and $\mathcal{F} \subseteq \Sigma^*$ is not nonterminal bounded and hence in particular not linear context-free. But if R and $\mathcal{F}$ are not too complicated then the nonlinearity is often just due to the fact that $L(R, \mathcal{F})$ is defined by a concatenation of languages. This is problematic since linear languages are not closed under concatenation. But if we have a linear context-free grammar G with an unambiguous turn position, we can insert a second linear

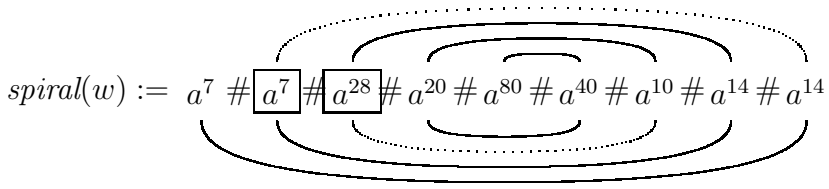


Figure 6.3: Linearisation of the word in Figure 6.1 on page 113.

language between the left and right portions of all the words in $L(G)$ and the resulting language will again be linear. Thus, provided the relation R is not too complicated we can often obtain a linear context-free language with a fixable turn position just by a permutation of the blocks in $L(R, \mathcal{F})$. The permutation *spiral* defined below can be computed recursively.

Definition 6.23 *Let $L := \Sigma^*(\#\Sigma^*\#\Sigma^*)^*$. Then the function $spiral : L \to L$ is defined by $spiral(w) := w$ for each $w \in \Sigma^*$ and $spiral(w_0\#w_1\#w) := w_0\#spiral(w)\#w_1$ for $w_0, w_1 \in \Sigma^*$ and $w \in L$.*

Note that each word in L is separated by the "#" symbols uniquely into an odd number of blocks. Hence, *spiral* is well defined and total. Moreover, it is easily seen to be a bijection on L.

The mapping *spiral* sorts one parity with ascending block numbers to the left and the other parity with descending block numbers to the right:

Example 6.24 *Let $n \in \mathbb{N}$, $w_i \in \Sigma^*$ for each $i \in [0, 2n]$. Then for the word $w := w_0\#w_1\#\cdots\#w_{2n}$ we have:*

$$spiral(w) = w_0\#w_2\#\cdots\#w_{2n-2}\#w_{2n}\#w_{2n-1}\#w_{2n-3}\#\cdots\#w_3\#w_1.$$

The mapping *spiral* permutes blocks in such a way that a line connecting neighbouring blocks from left to right is transformed into a spiral from the outside to the centre. One can see that by comparing Figure 6.1 on page 113 and Figure 6.3 on page 123. This behaviour of *spiral* motivates its name. Note that the correlation between blocks which are connected by a solid line on top of the word is from right to left, e.g., the second block in Figure 6.3 is connected with a dotted line to the rightmost block, since the left block is not twice as long as the right one. Instead it is the opposite way round. One can also see that all the correlations on top of a word and below the word can be checked by a single linear language, respectively.

As earlier mentioned, it is well known that each recursively enumerable set is the homomorphic image of the intersection of two context-free language. In the corresponding proofs the intersection of the two languages is essentially the set of valid computations of a Turing machine. At the beginning of Section 6.1 an informal description of the set of valid computations for an arbitrary Turing machine M has been provided. Let us denote this set by val_M. This characterisation of recursively enumerable sets has been strengthened in [3] by showing that it is sufficient to use *linear* context-free grammars. The essential idea is to show that *spiral*(val_M) is the intersection of two *linear* context-free languages. If the relation $R \subseteq \Sigma^* \times \Sigma^*$ is simple enough and $\mathcal{F} \subseteq \Sigma^*$ is a regular set then *spiral*$(L(R, \mathcal{F})$ is generated by a linear context-free grammar with unambiguous turn position whose ambiguity function is the canonical ambiguity of $L(R, \mathcal{F})$. The transition relation $\vdash_M$ of an arbitrary Turing machine M turns out to be "simple enough". But we still have to specify when a relation is simple enough.

6.6.3 Linear Block Correlation Languages

Definition 6.25 *A relation $R \subset \Sigma^* \times \Sigma^*$ is* simple *if there is an unambiguous context-free grammar $G = (N, \Sigma, P, S)$ such that $L(G) = \mathcal{L}(R)$ and $P \subseteq N \times (\Sigma^* N \Sigma^* \cup \{\#\})$.*

Definition 6.26 *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a simple relation and $\mathcal{F} \subseteq \Sigma^*$ a regular language. The* linear block correlation language *over the relation R and the set $\mathcal{F}$ is spiral$(L(R, \mathcal{F}))$. If spiral$(L(R, \mathcal{F}))$ is a block-correlation language then $\mathcal{F}$ is called the corresponding language of free blocks.*

Definition 6.27 *A* linear canonical grammar *for a linear block correlation language L over a relation $R \subseteq \Sigma^* \times \Sigma^*$ and a set $\mathcal{F} \subseteq \Sigma^*$ is a context-free grammar:*

$$G := \big(\{F\} \dot\cup N_{\mathcal{L}} \dot\cup N_{\mathcal{R}} \dot\cup N_{\mathcal{F}},\, \Sigma \cup \{\#\},\, P,\, S_{\mathcal{L}}\big),$$

where $G_{\mathcal{L}} := (N_{\mathcal{L}}, \Sigma \cup \{\#\}, P_{\mathcal{L}}, S_{\mathcal{L}})$, $G_{\mathcal{R}} := (N_{\mathcal{R}}, \Sigma \cup \{\#\}, P_{\mathcal{R}}, S_{\mathcal{R}})$, and $G_{\mathcal{F}} := (N_{\mathcal{F}}, \Sigma,\, P_{\mathcal{F}}, S_{\mathcal{F}})$ are unambiguous context-free grammars generating the languages $\mathcal{L}(R)$, $(\mathcal{L}(R))^R$, and $\mathcal{F}$, respectively. Moreover:

$$P_{\mathcal{L}} \subseteq N_{\mathcal{L}} \times (\Sigma^* N_{\mathcal{L}} \Sigma^* \cup \{\#\})$$
$$P_{\mathcal{R}} \subseteq N_{\mathcal{R}} \times (\Sigma^* N_{\mathcal{R}} \Sigma^* \cup \{\#\})$$
$$P_{\mathcal{F}} \subseteq N_{\mathcal{F}} \times \Sigma^* (N_{\mathcal{F}} \cup \{\varepsilon\})$$

Finally, the set of productions can be partitioned in four subsets:

$$\begin{aligned} P &:= (P_{\mathcal{L}} \cup P_{\mathcal{R}} \cup P_{\mathcal{F}}) \setminus (N_{\mathcal{L}} \cup N_{\mathcal{R}}) \times \{\#\} && \text{(i)} \\ &\cup \{A \to \#S_{\mathcal{L}}\# \mid A \to \# \in P_{\mathcal{L}}\} \cup \{A \to \#S_{\mathcal{R}}\# \mid A \to \# \in P_{\mathcal{R}}\} && \text{(ii)} \\ &\cup \{A \to u\#S_{\mathcal{R}} \mid \exists u \in \Sigma^* : A \to u \in P_{\mathcal{F}}\} && \text{(iii)} \\ &\cup \{S_{\mathcal{L}} \to S_{\mathcal{F}},\, F \to \varepsilon\} \cup \{A \to F\# \mid A \to \# \in P_{\mathcal{R}}\} && \text{(iv)} \end{aligned}$$

Now we check that each linear block correlation language has a linear canonical grammar. Since linear canonical grammars are defined over a simple relation a grammar $G_{\mathcal{L}}$ with the properties described in Definition 6.27 can be constructed for each linear block correlation language. An appropriate grammar $G_{\mathcal{R}}$ can be obtained by renaming the nonterminal belonging to $G_{\mathcal{L}}$ and reversing the right-hand side of each production. Finally, $G_{\mathcal{F}}$ which is right-linear exists since we require $\mathcal{F}$ to be regular for linear block correlation languages.

Now let G be a linear canonical grammar of some linear canonical block correlation language. Obviously, G is a linear context-free grammar. In the definition of P point (i) is the set of all productions of the three grammars $G_{\mathcal{L}}$, $G_{\mathcal{R}}$, and $G_{\mathcal{F}}$ which have at least one nonterminal on the right hand side. The productions of the form (ii) allow to insert a block pair at the position where the "#" symbol should be produced. Hence, instead of concatenating block pairs we insert them. The productions in (iii) and (iv) allow to switch from the $G_{\mathcal{L}}$ to $G_{\mathcal{F}}$ and from $G_{\mathcal{F}}$ to $G_{\mathcal{R}}$. Moreover, $S_{\mathcal{R}} \to F\#$, $F \to \varepsilon$ guarantee that G has erasing termination. Furthermore, G has unambiguous turn position. The left portion of each word is the longest prefix of the word which contain half of the "#" symbols.

With a lengthy but straightforward proof one can see that a linear canonical grammar generates the corresponding linear block correlation language.

It is possible to adapt all our proofs for block correlation languages and prove analogous facts for linear block correlation languages. Typically we just have to apply the *spiral* mapping to our constructions and adjust our argumentation accordingly. Except for the proof of Lemma 6.16 the adjustments are completely straightforward but somewhat awkward to write down. Let $R \subseteq \Sigma^* \times \Sigma^*$ be a simple relation and $\mathcal{F} \subseteq \Sigma^*$ a regular set. Furthermore, let G be a canonical grammar for the block correlation language over R and $\mathcal{F}$ and let G_{spiral} be a linear canonical grammar over R and $\mathcal{F}$. Then for each word $w \in \Sigma^*$ we have:

$$d_G(w) = d_{G_{spiral}}(spiral(w)).$$

This is due to the fact that a fixed permutation of blocks cannot change the number of choices for the position of the free block. The candidates for a

free block are those blocks to the left of the inner end of the spiral drawn in Fig 6.3 which have only solid lines below pairs which lie closer to the outside of the spiral and which have only solid lines on top of pairs which are close to the middle. In the proof of Theorem 6.10 the cancellation of the leftmost and rightmost pairs of blocks translates into cancellations of the outermost and innermost pairs of blocks, respectively. Since the transition relation $\vdash_M$ of an arbitrary Turing machine is simple we can also do our Turing machine construction. There is only one obstacle in the proof of Lemma 6.16. There we have used configurations of the form $q_0 q_0' n\$n^R$ as the free block language, which form a non-regular set. But linear block correlation languages only allow regular languages for the free block language. We can solve this problem by the use of a regular language $\mathcal{F}'$ which contains the prefixes of $\mathcal{F}$ until the \$ symbol. Fortunately the erased segment is redundant. It is easy to modify the Turing machine M such that it starts by appending a reversed copy of the integer which is found to the left of the \$ symbol. Then M works as described in the proof of Lemma 6.16. But before reentering the state q_0 the second segment is erased. The remaining statements can be transformed in a straightforward way. Thus, we get:

Theorem 6.28 *If f is a computable divergent total non-decreasing function then there is a cycle-free linear context-free grammar G with unambiguous turn position such that $\hat{d}_G$ is a divergent function satisfying $\hat{d}_G(n) \leq f(n)$ for all $n \in \mathbb{N}$.*

6.7 Rational Trace Language Generation

In the introduction of [5] the authors observe that the results on sublinear ambiguity in [33] can be transferred to the ambiguity of the generation of rational trace languages. This can also be done for the stronger results of this chapter. This section introduces trace languages briefly and explains informally how the results are transferred.

6.7.1 Preliminaries

We essentially adopt the definitions of [6, Chapter 5]. For the sake of self containedness we repeat them here with slight notational differences. Note that we use the word "ambiguity" instead of "multiplicity". An *independence alphabet* is a pair $(\Sigma, \mathcal{I})$ such that Σ is a finite alphabet and $\mathcal{I} \subseteq \Sigma^2$ is a symmetric and irreflexive relation on Σ. The *trace monoid* $\mathcal{M}(\Sigma, \mathcal{I})$ defined by an independence alphabet $(\Sigma, \mathcal{I})$ is the quotient $\Sigma^*/_{\equiv_\mathcal{I}}$ of the free monoid Σ^* with respect to the smallest congruence $\equiv_\mathcal{I}$ such that $ab \equiv_\mathcal{I} ba$ for each

pair $(a, b) \in \mathcal{I}$. A *trace language* is a subset of a trace monoid. Let $\varphi : \Sigma^* \to \mathcal{M}(\Sigma, \mathcal{I})$ be the canonical homomorphism. For each $w \in \Sigma^*$ we denote $\varphi(w)$ by $[w]_{\mathcal{I}}$. For a language $L \subseteq \Sigma^*$ we define $[L]_{\mathcal{I}} := \{[w]_{\mathcal{I}} \mid w \in L\}$. We say that a trace language $T \subseteq \mathcal{M}(\Sigma, \mathcal{I})$ is *generated* by a language $L \subseteq \Sigma^*$ if $T = [L]_{\mathcal{I}}$. A trace language is *rational* if it is generated by some regular set.

Let $M := \mathcal{M}(\Sigma, \mathcal{I})$ for some independence alphabet $(\Sigma, \mathcal{I})$ and let $L \subseteq \Sigma^*$. The *ambiguity power series* of L with respect to the trace monoid M is the mapping $d_{L,M} : \mathcal{M}(\Sigma, \mathcal{I}) \to \mathbb{N}$ defined by $d_{L,M}(t) := |\{w \in L \mid t = [w]_{\mathcal{I}}\}|$. The *ambiguity function* of L with respect to the trace monoid M is the mapping $\hat{d}_{L,M} : \mathbb{N} \to \mathbb{N}$ defined by $\hat{d}_{L,M}(n) := \max\{d_{L,M}(t) \mid t \in \mathcal{M}(\Sigma, \mathcal{I}) \text{ and } |t| = n\}$.

6.7.2 Sublogarithmic Ambiguity of Trace Language Generation

In Section 6.6 we have shown that we can obtain very slowly growing infinite ambiguities even for linear context-free grammars with unambiguous turn position. That is we could strengthen Theorem 6.18 to Theorem 6.28. In this section we transfer the result to the ambiguity of a rational trace language generation.

Let us consider an arbitrary canonical linear grammar $G = (N, \Gamma, P, S)$ for a linear block correlation language over a simple relation $R \subseteq \Sigma^+ \times \Sigma^+$. Let $\bar{\Gamma} := \{\bar{a} \mid a \in \Gamma\}$ be a disjoint copy of the symbols in Γ. For a word $w \in \Gamma^*$ we define $\bar{w}$ in the natural way as $h(w)$, where $h : \Gamma^* \to \bar{\Gamma}^*$ is the homomorphism defined by $h(a) := \bar{a}$ for each $a \in \Gamma$. Let $G' := (N, \Gamma \cup \bar{\Gamma}, P', S)$ where $P' := \{A \to u\bar{v}^R B \mid A \to uBv \in P\} \cup (P \cap (N \times \{\varepsilon\}))$. Obviously, $L(G')$ is regular. By definition no block can ever be empty since $R \subseteq \Sigma^+ \times \Sigma^+$. The words of $L(G')$ can be divided in blocks separated by the string $\#\bar{\#}$. But then $L(G')$ contains only one block which is generated without interleaved symbols from $\bar{\Sigma}$. This block corresponds to the free block of G. Therefore, G' is unambiguous which implies that $\hat{d}_G = \hat{d}_{L,M}$, where $M := \mathcal{M}(\Gamma, \mathcal{I})$ and $\mathcal{I} := \Gamma \times \bar{\Gamma} \cup \bar{\Gamma} \times \Gamma$. Since Turing machine configurations contain at least one symbol our construction does not require empty blocks. Therefore, we obtain:

Theorem 6.29 *If f is a computable divergent total non-decreasing function then there is a regular language $L \subseteq (\Sigma \cup \bar{\Sigma})$ where $\bar{\Sigma}$ is a disjoint copy of Σ, $\mathcal{I} := \Sigma \times \bar{\Sigma} \cup \bar{\Sigma} \times \Sigma$, and $M := \mathcal{M}(\Sigma, \mathcal{I})$ such that $\hat{d}_{L,M}$ is a divergent function satisfying $\hat{d}_{L,M}(n) \leq f(n)$ for all $n \in \mathbb{N}$.*

6.8 Conclusion

We have seen that for each computable divergent total non-decreasing function there is an inherent ambiguity function which falls below f. We have not examined which functions are indeed ambiguity functions. Seemingly there are no substantial gaps below linear ambiguity, in contrast to the gap between exponential and polynomially bounded ambiguity. But how can we characterise the "density" of ambiguity functions formally? To examine this question one could improve the estimation in this chapter. There is no need to use single steps of Turing machines as a means of computation. Instead we can allow unambiguous context-free relations to perform single steps. Clearly for the computational power this is unimportant but it provides more control over the length of the computations, which is crucial to control the ambiguity in our construction.

By the results of this chapter it is obvious that for each context-free grammar G_1 with a divergent ambiguity function we can find another context-free grammar G_2 with a divergent ambiguity function such that for any $c \in \mathbb{N}$ we have $d_{G_1}(n) \geq d_{G_2}(cn)$ for all but finitely many $n \in \mathbb{N}$. A natural question is whether there is a context-free language L such that a similar property holds for all the context-free grammars generating L. If this is the case L would not have an inherent ambiguity function. Are there context-free languages which do not have an inherent ambiguity function?

Chapter 7

The Gap Theorem

7.1 Introduction

In this chapter we will see that there is a gap between exponential and polynomially bounded ambiguity. That is, each ambiguity function of a context-free grammar is either in $\mathcal{O}(n^k)$ for some $k \in \mathbb{N}$ or in $\Omega^T(2^n)$ $(= 2^{\Omega(n)})$. For instance a function in $\Theta(2^{\sqrt{n}})$ cannot be an ambiguity function for any context-free grammar. For cycle-free context-free grammars we even obtain an overall upper bound of $\mathcal{O}^T(2^n)$. A similar gap has recently been found for census functions, which counts the number of word of a given length [18]. The gap for ambiguity functions is due to fundamental structural differences between polynomially bounded and exponentially ambiguous context-free grammars. These differences are expressed by two intuitive but undecidable necessary and sufficient criteria separating the classes of context-free grammars with polynomially bounded ambiguity and with exponential ambiguity. One criterion states that a context-free grammar G can be at most polynomially ambiguous if we can obtain an unambiguous grammar G' by the insertion of markers into the right-hand sides of bounded productions. The other criterion states that a context-free grammar is exponentially ambiguous if its set of pumping trees is ambiguous. Thus, each criterion is designed to prove a certain type of ambiguity when satisfied. For both criteria it is hard to figure out the consequences of their violation in terms of ambiguity. But we do not need technical and less insightful computations for the ambiguity of grammars which violate a criterion. Instead we will just prove that each cycle-free context-free grammar satisfies at least one of the two criteria. Clearly no cycle-free context-free grammar can satisfy both criteria since its ambiguity cannot be exponential and polynomially bounded at the same time. Hence, each cycle-free context-free grammar satisfies exactly one of the two criteria.

Thus, the two criteria turn out to be complementary. This fact implies the necessity of both criteria. Moreover, it proves a gap between polynomially bounded and exponential ambiguity. The necessity of the criterion for polynomially bounded ambiguity also implies that each language $L \in PCFL$, i.e., each context-free language with polynomially bounded ambiguity, is a bounded marker language. Since $BMCFL \subseteq PCFL$ has been shown in Observation 5.18 we obtain that the class of bounded marker languages and the class of context-free language with polynomially bounded ambiguity coincide, i.e. $BMCFL = PCFL$. Thus, the results obtained in Section 5.2 for bounded marker languages also hold for each context-free language with polynomially bounded ambiguity.

It is well known that ambiguity of a context-free grammar is undecidable. One might tend to believe that the task of deciding whether a context-free grammar is ambiguous or not becomes easier if we get the promise that it is extreme with respect to ambiguity, i.e., it is either unambiguous or exponentially ambiguous. But there is a subclass of provably extreme context-free grammars for which unambiguity is undecidable. However, for a given context-free grammar G, by an efficient algorithm, we can compute a constant $k_G \in \mathbb{N}$ such that the ambiguity-function of G is either exponential or in $\mathcal{O}(n^{k_G})$. We will see that the sufficient criterion for polynomially bounded ambiguity also gives rise to some closure properties and a good estimation of the polynomial degree of a context-free grammar with polynomially bounded ambiguity.

7.2 Sufficient Criterion for *ECFG*

In this section we prove that a cycle-free context-free grammar G is exponentially ambiguous if its set of pumping trees Λ_G is ambiguous. In other words, there are two different derivation trees with a common interface such that the frontier contains at least one node labelled with the same nonterminal as the root. If this criterion is satisfied we can pump up the ambiguity by concatenating these trees and choose randomly in each step the first or the second one. Thus, the number of combinations grows exponentially with respect to the length of the frontiers obtained by this method. This proof idea is rather intuitive but we have to take two technical problems into account:

- We have to prove that different combinations of different pumping trees with the same frontier never yield the same tree. For this purpose we show in Section 7.2.1 that the set of pumping trees with a fixed root is a free monoid over so-called prime pumping trees. In Section 7.2.2

we use these algebraic results to prove that two pumping trees with a common interface are the generators of a free submonoid.

- Pumping trees always contain a nonterminal in their frontier. But the ambiguity of a context-free grammar is defined by the number of derivation trees for *terminal* strings only. There are examples of unambiguous grammars (on the terminal strings) with exponential ambiguity in the sentential forms. In Section 7.6.1 we show that the ambiguity of sentential forms of pumping trees can be carried over to terminal strings for cycle-free context-free grammars without useless symbols. Therefore, the ambiguity of pumping trees is relevant for the ambiguity of the underlying context-free grammar.

The sufficient criterion for exponential ambiguity presented here is a generalisation of the decidable ambiguity criterion found in [20]. With our generalisation we lose decidability which will be shown in 7.5, but the criterion presented here turns out to be also necessary for exponential ambiguity which will be shown in 7.4. In fact, violation of this criterion leads to a polynomial upper bound for the ambiguity, which can be effectively constructed from the grammar.

Let $G = (N, \Sigma, P, S)$ be an arbitrary proper context-free grammar throughout this section.

7.2.1 The Free Monoid of Pumping Trees

An A-pumping tree for a nonterminal A is a derivation tree with the root A and exactly one leaf labelled A. We concatenate two A-pumping trees by identifying the root of the second one with the link of the first. This yields a pumping tree that inherits the root from the first and the link from the second pumping tree. Throughout section 7.2.1 we assume that A is a nonterminal of G. We now define pumping trees and their concatenation formally.

Definition 7.1 *The set of* A-pumping trees *is*

$$\Lambda_G^A := \{\rho \in \mathit{embedded}(\Delta_G) \mid A = \uparrow(\rho) \text{ and } \downarrow(\rho) \in \Sigma^* A \Sigma^*\}$$

An A-pumping tree $\vartheta \in \Lambda_G^A$ is said to be proper *if $\vartheta \neq A$.*

Definition 7.2 *Let $\vartheta_1, \vartheta_2 \in \Lambda_G^A$ be pumping trees. Then $\vartheta_1 = \tau_1 A \tau_1'$ and $\vartheta_2 = \tau_2 A \tau_2'$ for some $\tau_1, \tau_1', \tau_2, \tau_2' \in (P \cup \Sigma)^*$. The* concatenation $\odot$ *of ϑ_1 and ϑ_2 is defined by:*

$$\vartheta_1 \odot \vartheta_2 := \tau_1 \tau_2 A \tau_2' \tau_1'.$$

Note that τ_1, τ_1', τ_2, and τ_2' are uniquely determined in the definition above since $A \notin P \cup \Sigma$. The operation symbol $\odot$ is often omitted when we deal with A-pumping trees. However, the concatenation of pumping trees must not be confused with the concatenation of the corresponding tree strings.

Observation 7.3 *Let $\vartheta, \vartheta_1, \vartheta_2 \in \Lambda_G^A$. Then $|\vartheta_1 \odot \vartheta_2| = |\vartheta_1| + |\vartheta_2| - 1$. Moreover, $|\vartheta| = 1$ if and only if $\vartheta = A$.*

Theorem 7.4 *$(\Lambda_G^A, \odot)$ is a monoid with right cancellation.*[1]

Proof. Let $\vartheta_i = \tau_i A \tau_i'$ for $i \in \{1, 2, 3\}$ and $\vartheta = \tau A \tau'$ be A-pumping trees. Obviously, $\vartheta_{12} := \vartheta_1 \odot \vartheta_2 = \tau_1\tau_2 A \tau_2'\tau_1'$ has the root A and the property $\downarrow(\vartheta_{12}) \in \Sigma^*\uparrow(\vartheta_{12})\Sigma^*$. Moreover, ϑ_{12} inherits the property to be an embedded tree from ϑ_1 and ϑ_2. Thus, ϑ_{12} is an A-pumping tree. Moreover, A is the unit A-pumping tree, i.e., $A \odot \vartheta = \vartheta = \vartheta \odot A$. Moreover, $(\vartheta_1 \odot \vartheta_2) \odot \vartheta_3 = \tau_1\tau_2 A \tau_2'\tau_1' \odot \vartheta_3 = \tau_1\tau_2\tau_3 A \tau_3'\tau_2'\tau_1' = \vartheta_1 \odot \tau_2\tau_3 A \tau_3'\tau_2' = \vartheta_1 \odot (\vartheta_2 \odot \vartheta_3)$. Hence, $\odot$ is associative. Therefore, Λ_G^A is a monoid with the operation $\odot$ and the unit A. To show that the monoid has right cancellation we assume $\vartheta_1 \odot \vartheta = \vartheta_2 \odot \vartheta$. Then $\tau_1 \tau A \tau' \tau_1' = \tau_2 \tau A \tau' \tau_2'$. Since this string contains only one A we observe $\tau_1\tau = \tau_2\tau$ and $\tau'\tau_1' = \tau'\tau_2'$. Thus, by the cancellation rules on strings we get $\tau_1 = \tau_2$ and $\tau_1' = \tau_2'$ which implies $\vartheta_1 = \vartheta_2$.

Definition 7.5 *A pumping tree $\vartheta \in \Lambda_G^A \setminus \{A\}$ is prime if and only if it is not the product of two proper pumping trees. That is:*

$$\forall \vartheta_1, \vartheta_2 \in \Lambda_G^A \setminus \{A\} : \vartheta_1 \odot \vartheta_2 \neq \vartheta.$$

The set of prime pumping trees is denoted by λ_G^A.

Lemma 7.6 *Each $\vartheta \in \Lambda_G^A$ can be written as a finite product of prime A-pumping trees.*

Proof. Assume to the contrary that $\vartheta \in \Lambda_G^A$ is a shortest A-pumping tree which cannot be written as a product of prime pumping trees. Then clearly $\vartheta \notin \lambda_G^A$. Hence, $\vartheta = \vartheta_1 \odot \vartheta_2$ for some $\vartheta_1, \vartheta_2 \in \Lambda_G^A \setminus \lambda_G^A$. By Observation 7.3 we have $|\vartheta_1| > 1$ and $|\vartheta_2| > 1$ which again by Observation 7.3 implies $|\vartheta_1| < |\vartheta|$ and $|\vartheta_2| < |\vartheta|$. Hence, by the minimality of ϑ we obtain that ϑ_1

[1] It also has left cancellation, a fact we do not use in the sequel. To understand left cancellation assume $\vartheta \odot \vartheta_1 = \vartheta \odot \vartheta_2$, where ϑ_1, ϑ_2 and ϑ are defined as in the proof of Theorem 7.4. Then $\tau\tau_1 A \tau_1'\tau' = \tau\tau_2 A \tau_2'\tau'$ and $\vartheta_1 = \vartheta_2$ follows immediately by the cancellativity of strings.

and ϑ_2 both can be written as a finite product of prime A-pumping trees. But then ϑ can be written as the product of these products contradicting the assumption that ϑ cannot be written as a finite product of prime A-pumping trees. □

Lemma 7.7 *An A-pumping tree $\vartheta \in \Lambda_G^A \setminus \{A\}$ is prime if and only if it does not have A-pumping subtrees other than A and ϑ, i.e., subtree$(\{\vartheta\}) \cap \Lambda_G^A = \{A, \vartheta\}$.*

Proof. For the if portion of the statement we consider an A-pumping tree ϑ which is not prime. Then ϑ is the product of two A-pumping trees $\vartheta_1, \vartheta_2 \in \Lambda_G^A \setminus \{A\}$. By definition of the product $\odot$ the A-pumping tree ϑ_2 is a subtree of $\vartheta_1 \odot \vartheta_2$. Hence, a non prime A-pumping tree ϑ always has a subtree not in $\{A, \vartheta\}$. Therefore, an A-pumping tree ϑ' which does not have A-pumping subtrees other than A and ϑ' itself is prime. For the only if portion we consider an A-pumping tree ϑ with a proper subtree $\vartheta_2 \in \Lambda_G^A \setminus \{A\}$, i.e. $\vartheta_2 \notin \{A, \vartheta\}$. Then $\vartheta = \tau\vartheta_2\tau'$ for some $\tau, \tau' \in (P \cup \Sigma)^*$ such that $\tau\tau' \neq \varepsilon$. But then $\vartheta_1 := \tau\uparrow(\vartheta_2)\tau' = \tau A\tau' \in \Lambda_G^A \setminus \{A\}$ and $\vartheta = \vartheta_1 \odot \vartheta_2$. Thus, ϑ is not a prime A-pumping tree. Therefore, a prime A-pumping tree ϑ' cannot have A-pumping subtrees other than A and ϑ'. □

Theorem 7.8 *Each A-pumping tree $\chi \in \Lambda_G^A \setminus \lambda_G^A$ has a unique prime factorisation, that is, $\exists! k \in \mathbb{N} : \exists! \theta_1, \ldots, \theta_k \in \lambda_G^A : \chi = \theta_1 \odot \cdots \odot \theta_k$.*

Proof. Assume to the contrary that $\chi \in \Lambda_G^A$ is a shortest A-pumping tree which does not have a unique prime A-pumping tree factorisation. By Lemma 7.6 then the tree χ has at least two prime A-pumping tree factorisations. If the rightmost prime A-pumping trees θ_1 and θ_2 of two different prime A-pumping tree factorisations of χ would coincide then right cancellation of θ_1 would yield an A-pumping tree shorter than χ which does not have a unique prime A-pumping tree factorisation, contradicting the assumed minimality of χ. Hence, the rightmost prime A-pumping trees θ_1 and θ_2 of two different prime A-pumping tree factorisations of χ cannot coincide. Now θ_1 and θ_2 both are subtrees of χ containing the unique occurrence of A in χ. Therefore, their phrases are not independent, which implies that one of them is a subtree of the other one. But then, by Lemma 7.7, the larger one is not a prime A-pumping tree. □

From Theorem 7.4 and 7.8 we immediately obtain the following theorem.

Theorem 7.9 *Λ_G^A is a free monoid over λ_G^A.*

Thus, we can apply [16, Corollary of Theorem 1.3.3 and Theorem 1.3.4] and obtain:

Theorem 7.10 *Let $\vartheta_1, \vartheta_2 \in \Lambda_G^A$. Then*

(i) $\vartheta_1\vartheta_2 = \vartheta_2\vartheta_1 \Rightarrow \vartheta_1 = \vartheta^k$ *and* $\vartheta_2 = \vartheta^l$ *for some* $\vartheta \in \Lambda_G^A$ *and some* $k, l \in \mathbb{N}$.

(ii) $\vartheta_1\vartheta_2 \neq \vartheta_2\vartheta_1 \Rightarrow \{\vartheta_1, \vartheta_2\}^*$ *is a free submonoid of* Λ_G^A.

7.2.2 Free Submonoids

We show that two different pumping trees $\vartheta_1, \vartheta_2 \in \Lambda_G^A$ with a common frontier are the generators of a free submonoid. First we show that ϑ_1, ϑ_2 do not commute:

Lemma 7.11 *Let $\vartheta_1, \vartheta_2 \in \Lambda_G^A$. Then*

$$\downarrow(\vartheta_1) = \downarrow(\vartheta_2) \text{ and } \vartheta_1 \neq \vartheta_2 \quad \Rightarrow \quad \vartheta_1\vartheta_2 \neq \vartheta_2\vartheta_1$$

Proof. Assume $\downarrow(\vartheta_1) = \downarrow(\vartheta_2)$ and $\vartheta_1 \neq \vartheta_2$ and $\vartheta_1\vartheta_2 = \vartheta_2\vartheta_1$. Then by Theorem 7.10 we have $\vartheta_1 = \vartheta^k$ and $\vartheta_2 = \vartheta^l$ for some $\vartheta \in \Lambda_G^A$ and some $k, l \in \mathbb{N}_0$. Now $\vartheta_1 \neq \vartheta_2$ implies $\vartheta \neq A$ and $l \neq k$. For some $v, x \in \Sigma^*$ we have $\downarrow(\vartheta) = vAx$. Thus, $v^kAx^k = \downarrow(\vartheta_1) = \downarrow(\vartheta_2) = v^lAx^l$. Since $k \neq l$ this implies $v = x = \varepsilon$. Finally, $\vartheta \neq A$ and $\updownarrow(\vartheta) = [A, A]$ contradicts the cycle-freeness of G. Hence, $\downarrow(\vartheta_1) = \downarrow(\vartheta_2)$ and $\vartheta_1 \neq \vartheta_2$ implies $\vartheta_1\vartheta_2 \neq \vartheta_2\vartheta_1$ □

Notation 7.12 *For $\vartheta_1, \vartheta_2 \in \Lambda_G^A$ and $n \in \mathbb{N}$ the n-fold iteration of the pumping tree concatenation $\odot$ is denoted by $\{\vartheta_1, \vartheta_2\}^n$. It must not be confused with the n-fold concatenation of the strings in $\{\vartheta_1, \vartheta_2\}$ over T_G. The latter thing is never meant when we refer to pumping trees. Therefore, $\{\vartheta_1, \vartheta_2\}^n$ contains trees not forests consisting of n trees.*

As an immediate consequence of Lemma 7.11 and Theorem 7.10 we obtain

Corollary 7.13 *Let $\vartheta_1, \vartheta_2 \in \Lambda_G^A$; $n \in \mathbb{N}$; and $\downarrow(\vartheta_1) = \downarrow(\vartheta_2)$. Then*

$$\vartheta_1 \neq \vartheta_2 \quad \Rightarrow \quad \forall n \in \mathbb{N} : |\{\vartheta_1, \vartheta_2\}^n| = 2^n.$$

Note that for two different pumping trees $\vartheta_1, \vartheta_2 \in \Lambda_G^A$ with a common frontier w_1Aw_2 and $n \in \mathbb{N}$ each element of $\{\vartheta_1, \vartheta_2\}^n$ has the same frontier $w_1^nAw_2^n$. Moreover, the length of the frontier is linear in n. Therefore, Corollary 7.13 implies that ambiguous pumping tree sets are always exponentially ambiguous. Hence:

Observation 7.14 *The set of pumping trees Λ_G is extreme with respect to ambiguity, i.e., Λ_G is either unambiguous or exponentially ambiguous.*

7.2.3 The Pumping Tree Criterion

Note that exponential ambiguity of the set of embedded trees of a proper context-free grammar $\tilde{G}$ does not imply that $\tilde{G}$ is ambiguous.

Example 7.15 *Let* $\tilde{G} := S \to SaAA \mid \varepsilon, \quad A \to \varepsilon$. *For each* $i \in \mathbb{N}$ *we have* $am_{\tilde{G}}([S, S(aA)^i)] = 2^i$. *The corresponding set of embedded trees is:*

$$\{[S, SaAA]\}^i S\{aA[A, \varepsilon],\, a[A, \varepsilon]A\}^i \subseteq cut(\Lambda_{\tilde{G}})$$

Despite that, the set $cut(\Lambda_{\tilde{G}})$ *is exponentially ambiguous the grammar* $\tilde{G}$ *is unambiguous since* a^j *has the unique derivation tree:*

$$[S, SaAA]^j[S, \varepsilon](a[A, \varepsilon][A, \varepsilon])^j.$$

for each $j \in \mathbb{N}$.

By definition the ambiguity of a context-free grammar G is determined by the ambiguity of Δ_G. We have shown that Λ_G is exponentially ambiguous if it is ambiguous (Observation 7.14.) The previous example shows that exponential ambiguity of Δ_G is not a trivial consequence of the exponential ambiguity of Λ_G. In the sequel we will carefully examine when a sentential form α containing nonterminals is more ambiguous then any terminal word generated by α. It will turn out that this cannot happen for sentential forms which are the frontiers of pumping trees of a proper context-free grammar. This will complete the prove that a proper context-free grammar is exponentially ambiguous if its set of pumping trees is ambiguous.

We start with some technical lemmata on tree strings. It states that whenever we get the same tree by attaching a tree ρ to the nonterminal A in the frontier of two A-pumping trees $\vartheta_1, \vartheta_2 \in \Lambda_G$ then $\vartheta_1 = \vartheta_2$ follows, provided ρ has a non-empty frontier not beginning with A.

Lemma 7.16 *Let* Γ *be an alphabet and* $T := T_\Gamma$. *Moreover, let* $\tau_1, \tau_2, \tau_1', \tau_2' \in T^*$ *and* $\omega \in \Delta^A$ *for some* $A \in \Gamma$. *Then*

$$\begin{aligned} &\big(\tau_1\omega\tau_2 = \tau_1'\omega\tau_2' \text{ and } \downarrow(\tau_1 A\tau_2) = \downarrow(\tau_1' A\tau_2') \text{ and } \downarrow(\omega) \in (T \setminus \{A\})T^*\big) \\ \Rightarrow \quad &\tau_1 = \tau_1'. \end{aligned}$$

Proof. Assume $\tau_1 \neq \tau_1'$ holds while the left-hand side of the statement above is true. Since $\tau_1\omega\tau_2 = \tau_1'\omega\tau_2'$, either τ_1 is a prefix of τ_1' or vice versa. Without loss of generality we assume the first. By Lemma 2.28 the two occurrences of ω in $\tau_1\omega\tau_2$, starting at position $|\tau_1|+1$ and $|\tau_1'|+1$, respectively, do not overlap. Hence, $\tau_1' = \tau_1\omega\tau$ for some $\tau \in T^*$. Now $\downarrow(\tau_1 A\tau_2) \in \{\downarrow(\tau_1)\}\{A\}T^*$. On the other hand $\downarrow(\tau_1' A\tau_2') = \downarrow(\tau_1\omega\tau A\tau_2') \in \{\downarrow(\tau_1)\}\{\downarrow(\omega)\}T^*$. But $\{\downarrow(\tau_1)\}\{A\}T^*$ and $\{\downarrow(\tau_1)\}\{\downarrow(\omega)\}T^*$ are disjoint, since $\downarrow(\omega) \in (T \setminus \{A\})T^*$. This contradicts $\downarrow(\tau_1 A\tau_2) = \downarrow(\tau_1' A\tau_2')$. Therefore, our assumption is false. □

Now we show that attaching a tree with a non-empty purely terminal frontier to two different trees with the same frontier at the same position within the frontier yields different trees.

Lemma 7.17 *Let $G = (N, \Sigma, P, S)$ be a context-free grammar. Moreover, let τ_1,τ_2,τ_1',$\tau_2' \in T_G^*$, $A \in N$, and $\omega \in subtree(\Delta_G)$ such that $\downarrow(\omega) \in \Sigma^+$ and $\uparrow(\omega) = A$. Then*

$$\big(\tau_1 A \tau_2 \neq \tau_1' A \tau_2' \;\; and \;\; \downarrow(\tau_1 A \tau_2) = \downarrow(\tau_1' A \tau_2')\big) \quad \Rightarrow \quad \tau_1 \omega \tau_2 \neq \tau_1' \omega \tau_2'.$$

Proof. Assume to the contrary $\tau_1\omega\tau_2 = \tau_1'\omega\tau_2'$ and $\tau_1 A\tau_2 \neq \tau_1' A\tau_2'$ and $\downarrow(\tau_1 A\tau_2) = \downarrow(\tau_1' A\tau_2')$. Since $\omega \in \Delta^A_{N\cup\Sigma}$ and $\downarrow(\omega) \in (T \setminus \{A\})T^*_{N\cup\Sigma}$, we obtain $\tau_1 = \tau_1'$ by Lemma 7.16. Finally, cancellation of $\tau_1\omega$ from the equation $\tau_1\omega\tau_2 = \tau_1'\omega\tau_2'$ leads to $\tau_2 = \tau_2'$, contradicting $\tau_1 A\tau_2 \neq \tau_1' A\tau_2'$. □

Now we define the set of cut of derivation trees not containing void symbols in their frontier. Formally we define:

Definition 7.18

$$\tilde{\Delta}_G := \{\rho \in cut(\Delta_G) \mid \downarrow(\rho) \; does \; not \; contain \; void \; nonterminals.\}$$

Lemma 7.19 *There is a homomorphism h on T_G^* such that h maps $\tilde{\Delta}_G$ injectively to Δ_G.*

Proof. For each non void $A \in N$ we choose a tree $\rho_A \in subtree(\Delta_G)$ such that $\updownarrow(\rho_A) \in \{A \times \Sigma^+\}$. Such a tree exists since G is reduced and A is not void. Using Lemma 7.17 we can show the injectivity of h restricted to $\tilde{\Delta}_G$ by an induction on the number of nonterminals in the frontier of the considered trees in $\tilde{\Delta}_G$. □

Definition 7.20 *A nonterminal A is* void *if $\{u \in \Sigma^* \mid A \overset{*}{\rightarrow}_G u\} = \{\varepsilon\}$.*

In other words, a nonterminal is void if it cannot generate any other string then the empty word.

Lemma 7.21 *If $\neg U(\Lambda_G^A)$ then A is not a void symbol.*

Proof. If there are $\vartheta_1, \vartheta_2 \in \Lambda_G^A$ such that $\vartheta_1 \neq \vartheta_2$ and $\downarrow(\vartheta_1) = \downarrow(\vartheta_2)$ then A is not a void symbol, i.e., $\exists u \in \Sigma^+ : A \overset{*}{\rightarrow}_G u$.

Obviously, $\vartheta_1 \neq \vartheta_2$ implies $\vartheta_1 \neq A$ or $\vartheta_2 \neq A$. Without loss of generality we assume $\vartheta_1 \neq A$. Hence, $\downarrow(\vartheta_1) = w_1 A w_2$ for some $w_1, w_2 \in \Sigma^*$. Since G is cycle-free we observe $w_1 w_2 \neq \varepsilon$. Since G has no useless symbols there is a $\rho \in subtree(\Delta_G)$ such that $\uparrow(\rho) = A$. For some $\tau_1, \tau_2 \in T_G^*$ we have $\vartheta_1 = \tau_1 A \tau_2$. Then $\tau_1\rho\tau_2 \in subtree(\Delta_G)$ such that $\downarrow(\tau_1\rho\tau_2) = w_1\downarrow(\rho)w_2 \neq \varepsilon$. Therefore, A is not a void symbol. □

Theorem 7.22 *The grammar G is exponentially ambiguous if it has an ambiguous set of pumping trees, i.e.,*

$$\neg U(\Lambda_G) \Rightarrow G \in ECFG.$$

Proof. It is easily seen that $\hat{d}_G(n) \in 2^{\mathcal{O}(n)}$ for any cycle-free context-free grammar. (See footnote 2 in Section 3.2 for details.) It remains to show that $\hat{d}_G(n) \in \Omega^T(2^n)$.

If $\neg U(\Lambda_G)$ then there are $\vartheta_1, \vartheta_2 \in \Lambda_G^A$ such that $\vartheta_1 \neq \vartheta_2$ and $\downarrow(\vartheta_1) = \downarrow(\vartheta_2)$. Since A is not useless for some $\tau_1, \tau_2 \in T_G$ we have $\rho := \tau_1 A \tau_2 \in cut(\Delta_G)$ such that $\downarrow(\rho) \in \Sigma^* A \Sigma^*$. By Lemma 7.21 the nonterminal A is not void.

Now let $n \in \mathbb{N}$. By Corollary 7.13 we have $|\{\vartheta_1, \vartheta_2\}^n| = 2^n$. The set $\{\vartheta_1, \vartheta_2\}^n$ consists of A-pumping trees all having the same frontier. We define:

$$\bar{\Delta}_G := \left\{\tau_1 \tau_1' A \tau_2' \tau_2 \mid \tau_1' A \tau_2' \in \{\vartheta_1, \vartheta_2\}^n\right\}.$$

We observe that $|\bar{\Delta}_G| = 2^n$. Since the trees in $\bar{\Delta}_G$ only contain one nonterminal and this nonterminal is not void we have $\bar{\Delta}_G \subseteq \tilde{\Delta}_G$. According to Lemma 7.19 there is a homomorphism h on T_G^* such that $h(\bar{\Delta}_G) \subseteq \Delta_G$ and $|h(\bar{\Delta}_G)| = |\bar{\Delta}_G| = 2^n$. Moreover, all the trees in $h(\bar{\Delta}_G)$ have the same purely terminal frontier with a length of $\Theta(n)$. This implies $\hat{d}_G(n) \in \Omega^T(2^n)$. □

7.3 Sufficient Criterion for *PCFG*

Definition 7.23 *Let $G = (N, \Sigma, P, S)$ be a proper context-free grammar. Let P_ω and $P_{<\omega}$ be the corresponding sets of unbounded and bounded productions, respectively. For a production $p = [A, u_0 A_1 u_1 \cdots A_{k-1} u_{k-1} A_k u_k] \in P$ where $k \in \mathbb{N}$, $A, A_i \in N$ and $u_0, u_i \in \Sigma^*$ for $i \in [1, k]$ we define the mapping $mark : P \to (N \times (N \cup \Sigma \cup P)^*)$ by:*

$$mark_G(p) := \begin{cases} [A, pu_0 A_1 pu_1 \cdots A_{k-1} pu_{k-1} A_k pu_k] & \text{if } p \in P_{<\omega} \\ p & \text{otherwise.} \end{cases}$$

The skeleton grammar *$s(G)$ of G is defined by:*

$$s(G) := (N, \Sigma \cup P_{<\omega}, mark(P), S).$$

Example 7.24

$$G := S \to AA, \quad A \to aAa \mid bAb \mid \varepsilon$$

The language $L(G)$ consists of all words over $\{a, b\}$ which are the result of the concatenation of two even length palindromes. The bounded productions are $P_{<\omega} := \{[S, AA],\ [A, \varepsilon]\}$. The production $[S, AA]$ occurs once in each derivation tree and $[A, \varepsilon]$ twice.

$$s(G) := S \to [S, AA]A[S, AA]A[S, AA], \quad A \to aAa \mid bAb \mid [A, \varepsilon].$$

Lemma 7.25 *$s(G) \in UCFG \Rightarrow \hat{d}_G(n) \in \mathcal{O}(n^k)$, for $k = \sup(L(s(G)))(P)$.*

Proof. The grammar $s(G)$ is obtained from $G = (N, \Sigma, P, S)$ by inserting bounded productions of G as terminal marker symbols. It is easily seen that the partition of the production set of a context-free grammar in bounded and unbounded productions is not affected by the insertion or cancellation of terminals in right-hand sides of productions. Hence, there is a maximal number of marker symbols which can occur in a word of $L' := L(s(G))$. This upper bound is $k := \sup(L')(P)$. Since L' is unambiguous and π_Σ is a bounded contraction of L' which yields $\pi_\Sigma(L') = L(G)$ we see that $L(G)$ is a bounded marker language with L' as a witness with marking constant k. According to Observation 5.18 this implies $\hat{d}_G(n) \in \mathcal{O}(n^k)$ □

The next example shows how Lemma 7.25 can be used.

Example 7.26 *We consider the grammar G of Example 7.24. It is easily seen that $s(G)$ is unambiguous. The production*

$$S \to [S, AA]A[S, AA]A[S, AA]$$

produces 3 markers and is used once while the production $[A, \varepsilon]$ produces one marker and is used twice. Therefore,

$$k := \sup(L(s(G)))(P) = \sup(L(s(G)))(P_{<\omega}) = 1 \cdot 3 + 2 \cdot 1 = 5.$$

Hence, by Lemma 7.25 we have $\hat{d}_G(n) \in \mathcal{O}(n^5)$.

The estimation of Example 7.26 is not sharp. In fact, it can be shown that $\hat{d}_G(n) \in \mathcal{O}(n)$ (see Example 7.53). The advantage of Lemma 7.25 is that it provides an upper bound for the polynomial degree of a grammar with polynomially bounded ambiguity, without a need to understand the generated language.

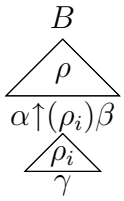


Figure 7.1: Expanding subtrees of pumping trees to pumping trees.

7.4 Gap Theorem

As an immediate consequence of the definitions we obtain:

Observation 7.27 *Let Γ be an alphabet. Let $L_1, L_2 \subseteq \Delta_\Gamma$ be tree languages. Then*

$$L_1 \subseteq L_2 \;\Rightarrow\; \big(U(L_2) \;\Rightarrow\; U(L_1)\big).$$

Definition 7.28 *A tree $\rho \in \mathit{embedded}(\Delta_G)$ is an unbounded production tree if it doesn't contain a bounded production and its frontier contains at most one nonterminal. Moreover, if the frontier contains a nonterminal then it belongs to the equivalence class of the root, i.e., the set of unbounded production trees is:*

$$\triangle_G := \{\rho \in \mathit{embedded}(\Delta_G) \mid \rho \in (P_\omega \cup \Sigma)^*([\uparrow(\rho)] \cup \{\varepsilon\})(P_\omega \cup \Sigma)^*\}.$$

Within Section 7.4 we assume that $G = (N, \Sigma, P, S)$ is an arbitrary proper context-free grammar.

Lemma 7.29 $U(\Lambda_G) \;\Rightarrow\; U\big(\mathit{subtree}(\Lambda_G)\big)$.

Proof. Assume $U(\Lambda_G)$ holds and $\rho_1, \rho_2 \in \mathit{subtree}(\Lambda_G)$ such that $\updownarrow(\rho_1) = \updownarrow(\rho_2)$. We have to show that $\rho_1 = \rho_2$ in this case. By definition there is a pumping tree $\tau_1\rho_1\tau_2 \in \Lambda_G$ for some $\tau_1, \tau_2 \in T_G^*$. Now $\tau_1\rho_1\tau_2 \overset{*}{\leftarrow} \tau_1{\uparrow}(\rho_1)\tau_2 = \tau_1{\uparrow}(\rho_2)\tau_2 \overset{*}{\rightarrow} \tau_1\rho_2\tau_2$ implies $\uparrow(\tau_1\rho_1\tau_2) = \uparrow(\tau_1\rho_2\tau_2)$ by Observation 2.16. On the other hand, $\downarrow(\tau_1\rho_1\tau_2) = \downarrow(\tau_1)\downarrow(\rho_1)\downarrow(\tau_2) = \downarrow(\tau_1)\downarrow(\rho_2)\downarrow(\tau_2) = \downarrow(\tau_1\rho_2\tau_2)$. Hence, $\tau_1\rho_1\tau_2$ and $\tau_1\rho_2\tau_2$ have the same interface. But then $\tau_1\rho_2\tau_2 \in \Lambda_G$. Therefore, $\tau_1\rho_1\tau_2 = \tau_1\rho_2\tau_2$ follows by $U(\Lambda_G)$. Finally, by cancellation we obtain $\rho_1 = \rho_2$ which completes the argument. Figure 7.1 illustrates the tree $\tau_1\rho_i\tau_2$ for an $i \in \{1, 2\}$, where $\alpha = \downarrow(\tau_1)$, $\beta = \downarrow(\tau_2)$, and $\gamma = \downarrow(\rho_i)$. Moreover $\alpha\gamma\beta \in \Sigma^*\{B\}\Sigma^*$. □

Note that the opposite implication $U(\Lambda_G) \Leftarrow U(\mathit{subtree}(\Lambda_G))$ also holds. This can be proved by the use of Observation 7.27 since $\Lambda_G \subseteq \mathit{subtree}(\Lambda_G)$. But we don't need this fact in the sequel.

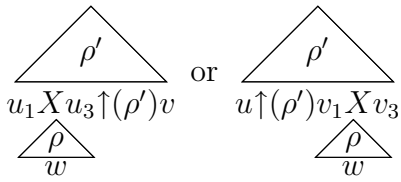


Figure 7.2: Expanding a tree only containing unbounded productions to a pumping tree.

Lemma 7.30 $\triangle\!\!\!\!\triangle_G \subseteq \mathit{subtree}(\Lambda_G)$.

Proof. Let $\rho \in \triangle\!\!\!\!\triangle_G$.
Case 1: The root $\uparrow(\rho)$ of ρ is not a bounded symbol. Then $X := \uparrow(\rho)$ is pumpable. Hence, by definition there is a derivation tree $\rho' \in \mathit{embedded}(\Delta_G)$ such that for $\alpha := X\uparrow(\rho')$ the relation $\vec{\alpha} \subseteq \vec{\rho'}$ holds, i.e., $\rho' \in \{\tau_1 X \tau_2 \uparrow(\rho')\tau_3, \tau_1\uparrow(\rho')\tau_2 X\tau_3\}$ for some $\tau_1, \tau_2, \tau_3 \in (P \cup \Sigma)^*$. In any case one occurrence of X can be replaced by ρ.[2] The resulting tree, either $\tau_1\rho\tau_2\uparrow(\rho')\tau_3$ or $\tau_1\uparrow(\rho')\tau_2\rho\tau_3$, is a pumping tree, since it has an occurrence of $\uparrow(\rho')$ in the frontier. Figure 7.2 illustrates the resulting tree.

Case 2: The root $\uparrow(\rho)$ of ρ is a bounded symbol. Let ν be a deepest node labelled with a symbol in $[\uparrow(\rho)]$, i.e., non of its descendants is in $[\uparrow(\rho)]$. (Obviously such a node exists, since the root node is labelled with a symbol in $[\uparrow(\rho)]$.) Assume ν is an internal node, i.e., $\rho[\nu] = p \in P$. By the choice of ν the production p is descending. Moreover, $\ell(p)$ is bounded since $\ell(p) \in [\uparrow(\rho)]$. Thus, p is a bounded production, which implies $\rho \notin \triangle\!\!\!\!\triangle_G$. This is a contradiction. Hence, our assumption is false and ν is a leaf. Therefore, $\vec{\rho} \cap [\uparrow(\rho)] \neq \emptyset$. Let $B \in \vec{\rho} \cap [\uparrow(\rho)]$. Then $B \vdash \uparrow(\rho)$. Therefore, there is a tree $\rho' = \tau_1\uparrow(\rho)\tau_2 \in \mathit{embedded}(\Delta)$ such that $\uparrow(\rho') = B$ and $\downarrow(\rho') \in \Sigma^*\uparrow(\rho)\Sigma^*$. Thus, $\tau_1\rho\tau_2 \in \Lambda_G$, i.e., $\rho \in \mathit{subtree}(\Lambda_G)$. □

As an immediate consequence of Lemma 7.29, Lemma 7.30, and Observation 7.27 we obtain:

Theorem 7.31 $U(\Lambda_G) \Rightarrow U(\triangle\!\!\!\!\triangle_G)$.

Obviously, $U(\triangle\!\!\!\!\triangle)$ and $\neg U(\triangle\!\!\!\!\triangle)$ are closed under deletion and insertion of terminals in bounded productions. Now Theorem 7.31 states that for the generation of polynomially bounded ambiguity, bounded productions are essential. If we insert sufficiently many markers in bounded productions to destroy their capacity to cause ambiguity then the resulting grammar should

[2]There may be more than one occurrence of X since $X = \uparrow(\rho)$ is not ruled out.

be unambiguous. This is exactly what we are about to do. Note that we will use productions of the original grammar as marker symbols in the constructed grammar.

Lemma 7.32 *Let $\rho \in subtree(\Delta_G)$. If ρ contains a node ν labelled with a bounded symbol A then each ancestor of ν is labelled with a bounded symbol.*

Proof. If ν' is an ancestor of ν labelled with B then B generates A (i.e. $B \vdash A$). Thus, by Lemma 4.12 we have $\sup_G(B) \leq \sup_G(A)$. Hence, B is a bounded symbol. □

Lemma 7.33 *Let $\rho \in subtree(\Delta_G)$. Let ν_1 and ν_2 be two nodes labelled with a bounded symbol where neither one is an ancestor of the other. Then there is a node $\nu < \min\{\nu_1, \nu_2\}$ such that the parse label of ν is a bounded production.*

Proof. Let ν be the first common ancestor of ν_1 and ν_2. Then $\nu \leq \min\{\nu_1, \nu_2\}$. Since neither one is an ancestor of the other $\nu < \min\{\nu_1, \nu_2\}$. By Lemma 7.32 the node ν and two distinct children of ν are labelled with bounded symbols. But among the descendants of a pumping production there is at most one bounded symbol. Therefore, the parse label of ν must be a descending production. Thus, we obtain that the parse label of ν is a bounded production. □

Lemma 7.34 *Each subtree ρ of a derivation tree which contains a bounded production (i.e., $\rho \in \{\mu \in subtree(\Delta_G) \mid \vec{\mu} \cap P_{<\omega} \neq \emptyset\}$) has a unique decomposition $\rho = \xi p \tau \chi$ where $p \in P_{<\omega}$ is a bounded production, such that $p\tau \in subtree(\Delta_G)$, $\xi\ell(p)\chi \in \triangle$, and $\uparrow(\xi\ell(p)\chi)$ is bounded.*

Proof. Let $\rho \in \{\mu \in subtree(\Delta_G) \mid \vec{\mu} \cap P_{<\omega} \neq \emptyset\}$. Then ρ contains at least one bounded production. For each internal node μ in ρ there is a uniquely defined decomposition $\rho = \xi p \tau \chi$ such that $|\xi| = \mu - 1$, $p\tau \in subtree(\Delta_G)$, and $\xi\ell(p)\chi \in embedded(\Delta_G)$. Our task is to find the appropriate μ. Since ξ cannot contain a bounded production but p is a bounded production, the only possible candidate for μ is the smallest integer i such that $\rho[i]$ is a bounded production. Now we choose the uniquely defined $\xi \in (N \cup \Sigma \cup P_\omega)^*$, $\tau, \chi \in (N \cup \Sigma \cup P)^*$, and $p \in P_{<\omega}$ with the property $\rho = \xi p \tau \chi$, $\rho' := \xi\ell(p)\chi$, and $p\tau \in subtree(\Delta_G)$. Obviously, $\rho' \in \{\rho'' \in embedded(\Delta_G) \mid \rho'' \in (P_\omega \cup \Sigma)^* N (P \cup \Sigma)^*\}$. Assume that there is a node μ' in the χ portion of ρ' whose parse label is a bounded production. Let $\mu := |\xi| + 1$. Since $\mu < \mu'$ the node μ' cannot be an ancestor of μ. On the other hand, μ is a leaf and is therefore no ancestor of μ'. Thus, by Lemma 7.33 there is a node $\nu < \mu$ whose parse label is a bounded production, which is a contradiction

to our choice of μ. That implies χ does not contain bounded productions. Therefore, $\rho' \in \{\mu \in \mathit{embedded}(\Delta_G) \mid \mu \in (P_\omega \cup \Sigma)^* N (P_{<\omega} \cup \Sigma)^*\}$. Assume $\ell(p) \notin [\uparrow(\rho')]$. Then there is a first ancestor ν of μ such that $\mu \notin [\nu]$. Since ν is an ancestor of μ we have $\nu < \mu$. The node μ is the leftmost node with a bounded production as its parse label. Hence, ν's parse label is not a bounded production. By Lemma 2.69 this implies that ν is labelled with an unbounded symbol or its parse label is a pumping production. Since ν is an ancestor of the node μ which has a bounded label, according to Lemma 7.32, the node ν is labelled by a bounded symbol. Hence, its parse label is a pumping production. By definition the child ν' of ν which is an ancestor of μ is labelled with a nonterminal B which is not equivalent to the label of ν. Therefore, according to Lemma 2.71 another child of ν is labelled with a nonterminal C in the equivalence class of the label of ν. Let D be the label of μ. Then $\{C, B\} \in \nabla_D$. Thus, D is pumpable. But then μ is not labelled with a bounded symbol according to Observation 2.78, which is a contradiction. Hence, the assumption is false and $\ell(p) \in [\uparrow(\rho')]$ follows. Thus, $\xi\ell(p)\chi \in \triangleq_G$ and $\uparrow(\xi\ell(p)\chi)$ is bounded. □

Lemma 7.35 *$U(\triangleq_G) \Rightarrow s(G) \in UCFG$.*

Proof. Observe that $p \in P$ is an unbounded production of G if and only if $\mathit{mark}_G(p)$ is an unbounded production of $s(G)$. Moreover, $P_\omega = P'_\omega$ which implies $\triangleq_G = \triangleq_{s(G)}$. For $U(\triangleq_G)$ we must show that arbitrary $\rho_1, \rho_2 \in \Delta_{s(G)}$ have common interfaces only if $\rho_1 = \rho_2$. We prove this by induction on $|\rho_1|_{P_{<\omega}}$. The basis is that $\rho_1 \in \triangleq_{s(G)}$. Now $\downarrow(\rho_1)$ does not contain any symbols in $P_{<\omega}$. Since each production in $h(P_{<\omega})$ generates symbols in $P_{<\omega}$ and $\updownarrow(\rho_1) = \updownarrow(\rho_2)$ we obtain that ρ_2 is in $\triangleq_{s(G)}$ too. By the observation above $\rho_1, \rho_2 \in \triangleq_G$. Hence, $U(\triangleq_G)$ implies $\rho_1 = \rho_2$. Assume the claim has been proved for all $\rho \in \Delta_{s(G)}$ with at most n bounded productions. Let ρ_1 contain $n+1$ bounded productions. By Lemma 7.34 for $i \in \{1, 2\}$ we can uniquely decompose $\rho_i = \xi_i\ h(p_i)\ \tau_i\chi_i$ such that $\rho'_i := \xi_i l(h(p_i))\chi_i \in \triangleq_{s(G)}$, $h(p_i) \in P'_{<\omega}$, and $h(p_i)\tau_i \in \Delta_{s(G)}$. Since all bounded productions happen to occur in $h(p_i)\tau_i$, it follows that p_1 and p_2 generate the leftmost and rightmost occurrences of symbols from $P_{<\omega}$ in $\downarrow(\rho_1)$ and $\downarrow(\rho_2)$, respectively. By $\updownarrow(\rho_1) = \updownarrow(\rho_2)$ this implies $p := p_1 = p_2$ and $\updownarrow(\rho'_1) = \updownarrow(\rho'_2)$. Now $\rho'_1, \rho'_2 \in \triangleq_{s(G)}$ implies $\rho'_1, \rho'_2 \in \triangleq_G$. Since $U(\triangleq_G)$ this implies $\rho'_1 = \rho'_2$. Now p is both a terminal of $s(G)$ and a bounded production of G. Thus, $h(p) = (A \to pu_0A_1pu_1 \cdots A_kpu_k)$ for some $k \in \mathbb{N}$, and for each $j \in \{1, \ldots, k\}$ we have $A, A_j \in N$ and $u_j \in \Sigma^*$. Then $\tau_i = \tau_{i,1} \cdots \tau_{i,k}$ has, for each $i \in \{1, 2\}$, a unique decomposition in k derivation trees $\tau_{i,1}, \ldots, \tau_{i,k} \in \Delta_{s(G)}$ such that $A_j = \uparrow(\tau_{i,j})$. Since $h(p)$ is a descending production it cannot occur in any

$\tau_{i,j}$. Hence, their yields cannot contain p. Therefore, we can uniquely retrieve the yield of each $\tau_{i,j}$ from $\downarrow(\rho_i)$, i.e., for each $j \in \{1,\ldots,k\}$ we have $\downarrow(\tau_{1,j}) = \downarrow(\tau_{2,j})$. Hence, $\updownarrow(\tau_{1,j}) = \updownarrow(\tau_{2,j})$ for each $j \in \{1,\ldots,k\}$. But since they do not contain $h(p)$ they contain at most n bounded productions. Therefore, by the inductive hypothesis $\tau_{1,j} = \tau_{2,j}$ for each $j \in \{1,\ldots,k\}$. This finally implies $\rho_1 = \rho_2$. □

Now we gather some previous results to show that we have several necessary and sufficient criteria for *PCFG*:

Theorem 7.36 *$G \in PCFG \Leftrightarrow G \notin ECFG \Leftrightarrow U(\Lambda_G) \Leftrightarrow U(\triangle_G) \Leftrightarrow s(G) \in UCFG$.*

Finally, we establish the gap between *ECFG* and *PCFG*

Proof. $G \in PCFG \overset{T7.22}{\Rightarrow} U(\Lambda_G) \overset{T7.31}{\Rightarrow} U(\triangle_G) \overset{L7.35}{\Rightarrow} s(G) \in UCFG \overset{L7.25}{\Rightarrow} G \in PCFG$. □

Theorem 7.37 *$\hat{d}_G = \Omega^T(2^n)$ or $\hat{d}_G = \mathcal{O}(n^k)$, where $k = \sup(L(s(G)))(P_{<\omega})$ and $P_{<\omega}$ is the set of G's bounded productions.*

Proof. If $\neg U(\Lambda_G)$ then $G \in ECFG$ by Theorem 7.22, i.e., $\hat{d}_G = \Omega^T(2^n)$. If $U(\Lambda_G)$ then $s(G) \in UCFG$ follows by Theorem 7.36. Thus, by Lemma 7.25 we obtain $\hat{d}_G(n) = \mathcal{O}(n^k)$. □

Note that the value of k in the theorem above can be computed in polynomial time with respect to the size of the grammar.

Corollary 7.38 *Each cycle-free context-free grammar is either in ECFG or PCFG.*

Finally, we can establish a close relationship between *UCFG* and *PCFG*:

Theorem 7.39 *The class of languages with polynomially bounded ambiguity PCFL, the class of bounded marker languages BMCFL and the closure of unambiguous languages UCFL under bounded contraction coincide.*

Proof. By definition *BMCFL* is a subset of the closure of *UCFL* under bounded contractions. By Corollary 4.6 the closure of *UCFL* under bounded contractions is a subset of *PCFL*. To see that $PCFL \subseteq BMCFL$ we take an arbitrary $L \in PCFL$ and show that $L \in BMCFL$ holds. Since $L \in PCFL$ we can choose a context-free grammar $G = (N, \Sigma, P, S) \in PCFG$ such that $L = L(G)$. Then $G \in U(\triangle)$ holds by Theorem 7.36. By Lemma 7.35 this implies $L(s(G)) \in UCFL$. Obviously, $L = \pi_{N\cup\Sigma}(L(s(G)))$ and $\pi_{N\cup\Sigma}$ is a bounded contraction for $L(s(G))$. Hence, L is a bounded marker language. □

Since *PCFL* = *BMCFL* by Theorem 7.39 we can transfer the notions of witnesses and marking constants to languages in *PCFL*.

Moreover, we can restate the results for the parallel parsing of bounded marker languages, Theorem 5.19 and Theorem 5.21, for languages with polynomially bounded ambiguity:

Corollary 7.40 *The word problem of a language $L \in PCFL$ with the marking constant m_L can be solved by a CREW-PRAM with $\mathcal{O}(n^{6+m_L})$ processors in time $\mathcal{O}(\log(n))$.*

Corollary 7.41 *Let $L \in PCFL$ and L which has a witness L' having the marking constant m. Moreover, let there be an unambiguous and linear context-free grammar G' with $L(G') = L'$. Then the word problem for L can be solved by a CREW-PRAM with $\mathcal{O}(n^{2+m})$ processors in time $\mathcal{O}(\log(n))$.*

Note that for each $G \in PCFL$ the marking constant of $L(G)$ cannot be higher than $\sup(L(s(G))(P)$. Thus, provided we know a context-free grammar which generates L with polynomially bounded ambiguity we can efficiently compute an upper bound for the marking constant, which is at the same time an upper bound for the degree of a polynomial which bounds the ambiguity of G.

7.5 Undecidability of *PCFG*

Definition 7.42 *A cycle-free context-free grammar is* extreme *with respect to its ambiguity if it is unambiguous or exponentially ambiguous.*

Theorem 7.43 *There is a class of cycle-free linear context-free grammars with unambiguous turn position which are provably extreme with respect to their ambiguity, but for which it is not decidable whether they are ambiguous.*

Proof. Let $\Gamma := \{A, B, C, S\}$ and $\Sigma := \{a, b, c, d, \#\}$ be alphabets. Let $x_1, \ldots, x_n,\ y_1, \ldots y_n \in \{a, b\}^+$. And let

$$\begin{aligned}
P_1 &:= \{S \to A,\ S \to B\} \\
&\quad \cup \{A \to x_j A d c^j \mid 1 \leq j \leq n\} \cup \{B \to y_j B d c^j \mid 1 \leq j \leq n\}, \\
P_2 &:= \{A \to x_j d c^j \mid 1 \leq j \leq n\} \cup \{B \to y_j d c^j \mid 1 \leq j \leq n\}, \\
\bar{P}_2 &:= \{A \to x_j C d c^j \mid 1 \leq j \leq n\} \cup \{B \to y_j C d c^j \mid 1 \leq j \leq n\} \\
&\quad \cup \{C \to \varepsilon\}.
\end{aligned}$$

In [16, Theorem 8.4.5] it is shown that the grammar $G_1 := (\{S, A, B\}, \Sigma, P_1 \cup P_2, S)$ is ambiguous if and only if the instance $((x_1, \ldots, x_n),(y_1, \ldots y_n))$ of the

Post Correspondence Problem has a solution.[3] The same argument applies for the grammar $G_2 := (\{S, A, B, C\}, \Sigma, P_1 \cup \bar{P}_2, S)$ which has erasing termination (see Definition 6.19). Now consider $G_3 = (\{S, A, B, C\}, \Sigma, P_1 \cup \bar{P}_2 \cup \{C \to \#S\#\}, S)$. It is easily seen that the production $C \to \#S\#$ does not introduce ambiguity. But if G_1 is already ambiguous then $\neg U(\Lambda_{G_3})$ holds. Hence, G_3 is extreme, but it is not decidable whether it is unambiguous or exponentially ambiguous. □

According to Theorem 7.36, in order to witness exponential ambiguity one only has to find two different pumping trees with a common interface. For an arbitrary context-free grammar G let $\langle G \rangle$ denote a reasonably simple encoding of G. For instance use the encoding in [17, Chapter 8.1, page 178]. Then we have

Theorem 7.44 *The set* $\{\langle G \rangle \mid G \in ECFL\}$ *is recursively enumerable.*

Proof. Construct a nondeterministic Turing machine which guesses two pumping trees and checks whether they are different and have a common interface. Accept if this is the case. Reject otherwise. □

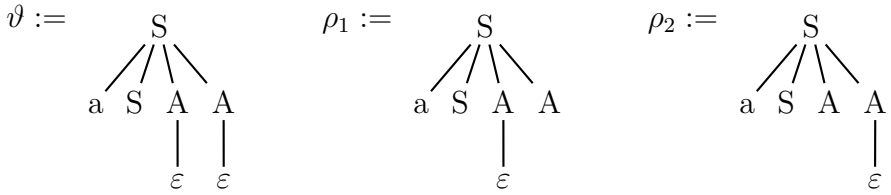
Another way to proof the previous theorem would be to guess an interface with the appropriate form for a pumping tree and check whether their are at least to pumping trees with this interface.

To check whether a context-free grammar G is exponentially ambiguous a deterministic simulation of the latter approach seems to be more efficient. Further heuristics to prove exponential ambiguity are discussed in the next section.

7.6 Heuristics to Prove Exponential Ambiguity

We have seen that a proper context-free grammar is exponentially ambiguous if and only if it has two different pumping trees with a common interface. We also know that this criterion is undecidable. As usual this problems is not hard to decide for each individual instance. There are some heuristics which can help to examine special cases. In section 7.6.1 we consider cuts of pumping trees, which can often lead to smaller witnesses for exponential ambiguity. In Section 7.6.2 we show that we can sometimes divide a grammar

[3] The undecidability of the ambiguity problem for context-free grammars is originally proved in [9, 8]



We have $\vartheta \in \Lambda_{\tilde{G}}$ and $\{\rho_1, \rho_2\} \subseteq cut(\vartheta)$. Moreover $\updownarrow(\rho_1) = [S, aSA] = \updownarrow(\rho_2)$.

Figure 7.3: Two different cuts of a pumping tree with a common interface.

G in several smaller ones such that G is exponentially ambiguous if and only if one of the smaller grammars is exponentially ambiguous. In Section 7.6.3 some heuristics are provided which are helpful to find pumping trees to witness exponential ambiguity.

7.6.1 Cuts of Pumping Trees

A pair of different pumping trees with a common interface serves as a witness for $\neg U(\Lambda_G)$ and thus for exponential ambiguity. What about witnesses for $\neg U(cut(\Lambda_G))$? They are often by far smaller and therefore easier to find. But do they imply exponential ambiguity in general? The answer is no. To see that let us revisit the grammar of Example 7.15. In Figure 7.3 we see two different cuts ρ_1 and ρ_2 of a pumping tree with the common interface aSA. Despite their difference, they are both cuts of the same pumping tree ϑ. In fact the set $\Lambda_G = \{([S, aSAA]a)^i S[A, \varepsilon]^2 i \mid i \in \mathbb{N}.\}$ of pumping trees is unambiguous, while the set $cut(\Lambda_G)$ is exponentially ambiguous as shown in Example 7.15. The loss of ambiguity in Λ_G is due to the fact that A is a void nonterminal, i.e., a nonterminal which cannot generate something else then the empty word. Thus we cannot exploit pairs of cuts of pumping trees as witnesses of exponentially ambiguity in general. But we just need a very weak additional restriction to use them. With the help of Lemma 7.17 we can easily show

Theorem 7.45

$$\neg U(\{\rho \in cut(\Lambda_G) \mid \downarrow(\rho) \textit{ does not contain void symbols.}\}) \Rightarrow \neg U(\Lambda_G).$$

While ε-productions often help to design a context-free grammar, void nonterminals can always be eliminated from a grammar in a trivial way yielding a smaller equivalent grammar. Therefore, void nonterminals almost never occur in practical examples This makes the following corollary frequently helpful:

Corollary 7.46 $\neg U(cut(\Lambda_G)) \Rightarrow \neg U(\Lambda_G)$.

7.6.2 Divide and Conquer on Grammars

We have seen that a context-free grammar G is exponentially ambiguous if and only if the set of embedded trees without bounded productions is ambiguous (Theorem 7.36). Hence we can consider G without bounded productions. The elimination of bounded productions may cause a decomposition of the dependency graph in several connected components. Each component can be considered as a smaller grammar to examine. This division in several smaller context-free grammars may simplify the proof of exponential ambiguity.

Lemma 7.47 *Let $G = (N, \Sigma, P, S)$ be a proper context-free grammar and $N_r := \{B \in N \mid B = S \text{ or } \exists p \in P_{<\omega} : |r(p)|_B > 0\}$. Then $G \in PCFG$ if and only if for all $A \in N_r$ the context-free grammar $G_A := (N, \Sigma \cup \tilde{N}, P_\omega \cup P', A) \in UCFG$ is unambiguous (but not necessarily reduced), where $\tilde{N} := \{\tilde{X} \mid X \in N\}$ is a copy of the nonterminals such that $\tilde{N} \cap N = \emptyset$, and $P' := \{A \to \tilde{A} \mid A \in N\}$.*

Proof. If $G \notin PCFG$ then $U(\triangleq_G)$ is false. Thus, there are two different trees $\rho_1, \rho_2 \in \triangleq(G)$ with $\updownarrow(\rho_1) = \updownarrow(\rho_2)$. By the definition of N_r there must be some $A \in N_r$ such that $A \vdash \uparrow(\rho_1)$ and all the productions applied to ρ_1 and ρ_2 are in the production set of G_A. Hence, $\rho_1, \rho_2 \in embedded(G_A)$. If ρ_1 and ρ_2 still contains nonterminals we can replace them by the use of productions in P'. This yields two derivation trees ρ_1' and ρ_2'. Since ρ_1 and ρ_2 do not contain productions in P' we can retrieve ρ_1 and ρ_2 from ρ_1' and ρ_2', respectively. Hence, $\rho_1' \neq \rho_2'$ holds which implies that G_A is ambiguous.

On the other hand, if G_A is ambiguous for some $A \in N_r$ then we can find two different derivation trees ρ_1 and ρ_2 which generate the same word. After cutting off all the subtrees of the form $[A, \tilde{A}]\tilde{A}$, where $A \in N_r$, we obtain two different trees ρ_1' and ρ_2' which still have a common frontier. Moreover, they both lie in $\triangleq_G$. Thus, $U(\triangleq_G)$ is false which implies that $G \notin PCFG$.

□

The previous lemma splits a context-free grammar $G = (N, \Sigma, P, S)$ into $|N_r|$ many grammars. In fact we can often consider a smaller number of grammars. It can be easily seen that it is sufficient to consider the grammars with start symbols in $\{A \in N_r \mid \forall B \in N_r : \neg(B \vdash_{G_B} A\}$. To see that observe that for $A, B \in N_r$ such that $B \vdash_{G_B} A$ the grammar G_B has all the derivation trees of G_A as embedded trees. Hence the examination of G_B already contains an examination of G_A.

7.6.3 Other Heuristics

The following sufficient criterion for $\neg U(\Lambda_G)$ depends only on the interfaces of two pumping trees. It is not necessary to consider their internal structure.

Corollary 7.48 *Let $G = (N, \Sigma, P, S)$ be a proper context-free grammar.*

$$\left.\begin{array}{c}\exists \vartheta_1, \vartheta_2 \in \Lambda_G^A;\ v, x \in \Sigma^+;\ k, l, m, n \in \mathbb{N}\ : \\ \downarrow(\vartheta_1) = v^k A x^l\ \wedge\ \downarrow(\vartheta_2) = v^m A x^n\ \wedge\ lm \neq kn\end{array}\right\} \Rightarrow\ \neg U(\Lambda_G)$$

Proof. If the left-hand side is satisfied then $\downarrow(\vartheta_1\vartheta_2) = v^{k+m}Ax^{l+n} = \downarrow(\vartheta_2\vartheta_1)$. Assume $\vartheta_1\vartheta_2 = \vartheta_2\vartheta_1$. By Theorem 7.10 then $\vartheta_1 = \vartheta^p$ and $\vartheta_2 = \vartheta^q$ for some $p, q \in \mathbb{N}$ where $\downarrow(\vartheta) = \bar{v}A\bar{x}$ for some $\bar{v}, \bar{x} \in \Sigma^+$. This implies $k \cdot |v| = p \cdot |\bar{v}|\ \wedge\ l \cdot |x| = p \cdot |\bar{x}|\ \wedge\ m \cdot |v| = q \cdot |\bar{v}|\ \wedge\ n \cdot |x| = q \cdot |\bar{x}|$. By multiplication we obtain $k \cdot |v| \cdot n \cdot |x| = p \cdot |\bar{v}| \cdot q \cdot |\bar{x}| = l \cdot |x| \cdot m \cdot |v|$. Since $|v| \cdot |x| > 0$ we obtain by division $kn = lm$ which is a contradiction. Hence, $\vartheta_1\vartheta_2 \neq \vartheta_2\vartheta_1$ and $\neg U(\Lambda_G)$ follows. □

Example 7.49 *Let $G := \big(S \to a^2Sb^3 \mid a^5Sb^7 \mid \varepsilon\big)$. Let $\vartheta_1 := [S, a^2Sb^3]a^2Sb^3$ and $\vartheta_2 := [S, a^5Sb^7]a^5Sb^7$. Then for $A := S$, $v := a$, $x := b$, $k = 2$, $l = 3$, $m = 5$, and $n = 7$ we have $\vartheta_1, \vartheta_2 \in \lambda_G^A$, $\downarrow(\vartheta_1) = v^kAx^l$ and $\downarrow(\vartheta_2) = v^mAx^n$. Moreover, $lm = 15 \neq 14 = kn$. Therefore, according to Corollary 7.48 the grammar G is exponentially ambiguous.*

Even though Corollary 7.48 is a quite elementary consequence of Theorem 7.22 it is useful to have it in mind when searching for a proof for exponential ambiguity since the repetitive pattern of the frontiers of ϑ_1 and ϑ_2 is easily observed by a human. Then one has to verify the inequation $lm \neq kn$. If the considered grammar is exponentially ambiguous it is quite likely that the inequation holds by chance for the first considered pair of pumping trees.

Finally, we prove a decidable sufficient criterion for exponential ambiguity, which is a proper generalisation of the ambiguity criterion presented in [20] even for grammars in Chomsky normal form.

Corollary 7.50 *A proper context-free grammar $G = (N, \Sigma, P, S)$ is exponentially ambiguous if it contains a nonterminal which is left- as well as right recursive. Formally that is:*

$$\big(\exists A \in N;\ v, x \in \Sigma^+ : A \overset{+}{\to}_G vA\ \wedge\ A \overset{+}{\to}_G Ax\big)\ \Rightarrow\ \neg U(\Lambda_G).$$

Proof. If G is cycle-free and $A \in N$ is as well left- and right recursive, then there are $\vartheta_1, \vartheta_2 \in \Lambda_G^A$ such that $\downarrow(\vartheta_1) = vA$ and $\downarrow(\vartheta_2) = Ax$ for some $v, x \in \Sigma^+$. Thus, for $k := n := 1$ and $l := m := 0$ we have $\downarrow(\vartheta_1) = v^kAx^l$, $\downarrow(\vartheta_2) = v^mAx^n$, and $lm = 0 \neq 1 = kn$. Therefore, by Corollary 7.48 we obtain $\neg U(\Lambda_G)$. □

Example 7.51 *Consider the following context-free grammars each of which generates the set of balanced parenthesis:*

$$G_1 := \big(S \to (S)S \mid \varepsilon\big) \quad G_2 := \big(S \to S(S)S \mid \varepsilon\big) \quad G_3 := \big(S \to SS \mid (S) \mid \varepsilon\big).$$

The grammars G_1 is easily seen to be unambiguous. For the grammar G_2 the start symbol S is as well left- and right recursive due to the single production $S \to S(S)S$. Similarly G_3 has a left- and right recursive start symbol due to the production $S \to SS$. Therefore, by Corollary 7.50 the grammars G_2 and G_3 are exponentially ambiguous.

Chomsky Normal Form examples which do not satisfy the criterion in [20] but which can be proved to be exponentially ambiguous by Corollary 7.50 are:

Example 7.52

$$\begin{aligned} G_1 &:= (S \to AS \mid SA, \quad A \to a) \\ G_2 &:= (S \to SA \mid BS \mid c, \quad A \to SA \mid a, \quad B \to BS \mid b) \end{aligned}$$

For both G_1 and G_2 the start symbol is obviously left- as well as right recursive. Therefore, they are exponentially ambiguous by Corollary 7.50. But neither G_1 nor G_2 satisfies the sufficient ambiguity criterion of [20] for different reasons: The grammar G_1 does not have any so-called "closed production set". Therefore, there is no closed production set with the required properties. The second grammar has several closed production sets, but none of them contains both productions $S \to SA$ and $S \to BS$ which leads to a violation of the criterion of [20]. A proof of this fact can be found in [31].

7.7 Estimation of Polynomial upper Bounds

While in general it is undecidable whether a context-free grammar is in $PCFG$ it might be decidable in many special cases. In case we know that a context-free grammar $G = (N, \Sigma, P, S)$ is in $PCFG$ we can compute a $k \in \mathbb{N}$ such that $\hat{d}_G \in \mathcal{O}(n^k)$. An appropriate candidate for k is the maximal number of marker symbols belonging to P which show up in words generated by the skeleton grammar $s(G)$,i.e., $\sup(L(s(G)))(P)$. Since there are grammars with sublinear ambiguity one cannot expect that this estimation is sharp. In fact, the marking of $s(G)$ is rather abundant in order to guarantee that we eliminate each possible ambiguity.

In many cases we can cancel markers from bounded productions of the grammar $s(G)$ and the resulting grammar is still unambiguous. According

to Lemma 4.5 the number of markers of such a modified grammar is still an upper bound for the degree of the polynomial which is an upper bound for the ambiguity of G. In fact, it is even not necessary that the modified grammar is unambiguous. A finite degree of ambiguity is sufficient.

There cannot be an algorithm which decides which markers of $s(G)$ are required and which are superfluous. Otherwise we would know whether we need at least one marker or not. This would mean to decide for a grammar $G \in PCFG$ whether it is unambiguous or not. It is well known that it is undecidable whether a general context-free grammar is unambiguous. Since the standard construction only handles grammars whose degree of ambiguity is bounded by 2, it is undecidable whether or not a context-free grammar $G \in PCFG$ is unambiguous.

Despite that there are several cases where we can save some markers. As usual one can invest a lot of effort to handle special cases of an undecidable problem and improve the algorithms an infinite number of times. Therefore, we do not go into detail here. But in the sequel we sketch one improvement without a proof of its correctness.

One can modify the grammar G such that each bounded nonterminal occurs at most once in a sentential form. To obtain that we can make copies of bounded nonterminals and their productions. Then the resulting grammar is marked. How this can be done in general is sketched as follows:

For each occurrence of a bounded symbol in the frontier of a sentential derivation tree we can move up the path of its ancestors. Each time we hit a node parse labelled with a bounded production we can denote this production and the number of the child where we came from. It can be shown that this procedure yields a history which cannot be equal for two different occurrences of a leaf labelled with a bounded nonterminal. Moreover, since we only consider bounded productions in the history its length is bounded by a constant only depending on G. This can be used to attach to each bounded symbol its history. The productions for these nonterminals with a history are the same as for the original nonterminal except for the fact that they propagate their history to those of their children which are labelled with bounded symbols. In case of a bounded production the history is appended in an appropriate way. The construction above yields a grammar G' with the same ambiguity power series as G with respect to words over terminals. If we have thus achieved a grammar with the same ambiguity as G such that each bounded nonterminal appears at most once in a sentential form then the cancellation of a single marker reduces the number of markers in a word at most by one symbol. This provides a better control of the marking process and may help to save some markers.

Now we revisit the grammar of Example 7.24 to demonstrate the im-

provements mentioned above:

Example 7.53 *Let G be the grammar of Example 7.24:*

$$G := S \to AA, \quad A \to aAa \mid bAb \mid \varepsilon$$

The corresponding skeleton grammar is:

$$s(G) := S \to [S, AA]A[S, AA]A[S, AA], \quad A \to aAa \mid bAb \mid [A, \varepsilon].$$

Note that $[S, AA]$ and $[A, \varepsilon]$ are terminal symbols of $s(G)$. We have seen in Example 7.26 that the grammar $s(G)$ leads to an estimation of $\hat{d}_G \in \mathcal{O}(n^5)$. For convenience we rename the marker symbols and obtain the grammar:

$$G_1 := S \to \#A\#A\#, \quad A \to aAa \mid bAb \mid \$.$$

Obviously, we can eliminate the first and last marker of the production $S \to \#A\#A\#$ without changing the ambiguity, since they always form the first and last symbol of the generated word. We obtain the unambiguous context-free grammar:

$$G_2 := S \to A\#A, \quad A \to aAa \mid bAb \mid \$.$$

The grammar G_2 improves the estimation to $\hat{d}_G \in \mathcal{O}(n^3)$. By inspecting the language we see that it is sufficient to know where the first palindrome ends and the second begins. Thus, we avoid to mark the middle of the palindromes. We obtain the unambiguous context-free grammar:

$$G_3 := S \to A\#A, \quad A \to aAa \mid bAb \mid \varepsilon.$$

The grammar G_3 improves the estimation to $\hat{d}_G \in \mathcal{O}(n)$. Even though we will not beat the estimation based on G_3 we consider another idea to show how we avoid bounded symbols which show up several times. We can also decide to mark the middle of the palindromes, while we drop the marker separating the two palindromes. This leads us from G_2 to the unambiguous context-free grammar G_4:

$$G_4 := S \to AA, \quad A \to aAa \mid bAb \mid \$$$

Obviously, with G_4 we obtain an estimation of $\hat{d}_G \in \mathcal{O}(n^2)$. Now we may discover that it is sufficient to mark the middle of only one palindrome. Knowing the middle of one palindrome the other palindrome and its middle are uniquely determined. But the production $A \to \$$ marks the middle of both palindromes.

We can avoid that by splitting the nonterminal A into one nonterminal A_1 for the generation of the first palindrome and one nonterminal A_2 for the second palindrome:

$$G' := S \to A_1A_2, \quad A_1 \to aA_1a \mid bA_1b \mid \varepsilon, \quad A_2 \to aA_2a \mid bA_2b \mid \varepsilon.$$

The index 1 of A_1 stores the information that A_1 is the left child of the production $S \to AA$. Similarly the index 2 of A_2 stores the information that we have entered the right child. We have not added the production $[S, AA]$ itself in the history of the nonterminals, because this is not necessary for this simple example. We also see how the history (here just the index 1 or 2) is propagated to the descendants. This attached history avoids two occurrences of the same bounded nonterminal in a single sentential form. Obviously, G' has the same ambiguity as G. Now we mark the grammar G':

$$s(G') := S \to \#A_1\#A_2\#, \quad A_1 \to aA_1a \mid bA_1b \mid \$_1, \quad A_2 \to aA_2a \mid bA_2b \mid \$_2.$$

We have already renamed $[S, A_1A_2]$ by $\#$, $[A_1, \varepsilon]$ by $\$_1$, and $[A_2, \varepsilon]$ by $\$_2$ Clearly the estimation of $\hat{d}_G$ by the use of $s(G')$ has not changed in comparison to the one based for $s(G)$. The advantage of $s(G')$ is that it allows to decide for each copy of the nonterminal A whether we want to keep the markers below. That is, we can decide for each palindrome separately whether we want to mark its middle, or not. This leads us to the unambiguous context-free grammars G_5 and G_6 which both yield the estimation $\hat{d}_G \in \mathcal{O}(n)$.

$$G_5 := S \to A_1A_2, \quad A_1 \to aA_1a \mid bA_1b \mid \varepsilon, \quad A_2 \to aA_2a \mid bA_2b \mid \$_2.$$

$$G_6 := S \to A_1A_2, \quad A_1 \to aA_1a \mid bA_1b \mid \$_1, \quad A_2 \to aA_2a \mid bA_2b \mid \varepsilon.$$

7.8 Grammar Parameters and Ambiguity

Two parameters of a context-free grammar are the maximum number of nonterminals on the right-hand side of a production and the total number of nonterminals. It is easier to obtain them than to compute Parikh suprema which require more insight into the structure of the productions.

Theorem 7.54 *Let $G = (N, \Sigma, P, S)$ be a context-free grammar. Let $m := |N|$ and $j := \max\{|r(p)|_N \mid p \in P\}$, i.e., m is the number of nonterminals and j the maximal number of nonterminals which occur on the right-hand side of a production. Then $\hat{d}_G \in \Omega^T(2^n) \cup \mathcal{O}(n^k)$, where $k = 2\frac{j^m-1}{j-1} - 1$.*

Proof. By Theorem 7.37 it is sufficient to show that $k \geq \sup(L(s(G)))(P_{<\omega})$, where $P_{<\omega}$ is the set of G's bounded productions. It is easily seen that the highest number of markers is reached by the context-free grammar:

$$G_{m,j} := (\{A_1, \ldots, A_m\}, \{a\}, P_{m,j}, A_m),$$

where $P_{m,j} := \{A_{i+1} \to A_i^j \mid i \in [1, \ldots, m-1]\} \cup \{A_1 \to a\}$. The grammar produces a single word with a unique derivation tree. For each $i \in [1, m]$ there are j^{m-i} many occurrences of the nonterminal A_i. If we consider $s(G)$ we see that the productions for the nonterminals labelled by A_1 only cause one marker each, while the others cause $j+1$-markers to occur. In total we obtain:

$$\begin{aligned} \sup(L(s(G)))(P_{<\omega}) &= j^{m-1} + (j+1) \cdot \sum_{i=1}^{m-1} j^{i-1} = j^{m-1} + \sum_{i=1}^{m-1} j^i + \sum_{i=1}^{m-1} j^{i-1} \\ &= 2j^{m-1} + 2\sum_{i=1}^{m-2} j^i + 1 = 2\sum_{i=0}^{m-1} j^i - 1 = k. \end{aligned}$$

□

In Theorem 7.54 we have seen an upper bound for the degree of the polynomial of a polynomially bounded grammar G which only depends on the number of nonterminals n and the number of nonterminal which can occur on the right hand side of a production j. But is this upper bound sharp, i.e., are there any context-free grammars which reach the polynomial upper bound provided by Theorem 7.54? We consider the case $n = j = 1$. Then the context-free grammar is linear and has one nonterminal only. Such a grammar is called a *minimal linear* grammar.

Yuji Kobayashi [21] provided the grammar G_{Kob} for the following lemma. He also sketched the corresponding proof:

Lemma 7.55 *The following minimal linear context-free grammar is $\Theta(n)$ ambiguous:*

$$G_{Kob} := S \to abSbaba \mid aSa \mid babaSab \mid b$$

Proof. Firstly we show that the following relationship holds:

$$\forall n \in \mathbb{N} : d_{G_{\text{Kob}}}(ab)^{3n+1}a = n+1.$$

To see that, we choose arbitrary $i, j \in \mathbb{N}$ such that $i + j = n$. (There are $n + 1$ many choices to do that.) Now we get:

$$\begin{aligned} S &\overset{i}{\Rightarrow} (ab)^i S (ba)^{2i} \\ &\Rightarrow (ab)^i a S a (ba)^{2i} \\ &\overset{j}{\Rightarrow} (ab)^i a (ba)^{2j} S (ab)^j a (ba)^{2i} \\ &\Rightarrow (ab)^i a (ba)^{2j} b (ab)^j a (ba)^{2i} \\ &= (ab)^i (ab)^{2j} ab (ab)^j (ab)^{2i} a \\ &= (ab)^{3n+1} a. \end{aligned}$$

Hence, $\hat{d}_{G_{\mathrm{Kob}}}(n) \in \Omega(n)$.

The grammar G_{Kob} is in *PCFG* if and only if $s(G_{\mathrm{Kob}})$ is unambiguous. The grammar $s(G_{\mathrm{Kob}})$ is obtained from G_{Kob} by replacing the production $S \to b$ by $S \to [S, b]b$. The string $[S, b]b$ just serves as a marker. We can replace it by # without changing the ambiguity. Hence, by Theorem 7.36 the grammar G_{Kob} is in *PCFG* if and only if the following grammar is unambiguous:

$$G := S \to \underbrace{abSbaba}_{1} \mid \underbrace{aSa}_{2} \mid \underbrace{babaSab}_{3} \mid \#.$$

Let $w \in L(G)$. From the left and right ends of w we move into the middle until we find two consecutive symbols which are equal. Then a vertical bar is drawn between these two symbols. Then we repeat the process recursively for the word between the bars until we hit the # symbol. This may yield for instance:

$$ab|baba|abababa|a|aba\#ababa|a|abababa|ab|baba.$$

A tree can be denoted by the sequence of applied productions, which is unique, since G is linear. Consecutive a's result when we switch from production 2 or 3 to production 1 or 2 in a derivation. Two consecutive b's are the result of a change from production 1 to 3. Note that consecutive terminals of the same type always emerge simultaneously on both sides of the start symbol (or the #-symbol in the last step). We proof the unambiguity of G by induction on the number of vertical bars which occur to the left of the #-symbol. Firstly we consider an arbitrary word $w \in L(G)$ without vertical bars. According to what we have said above each derivation of w is of one of the three forms 1^*23^*4, 1^+4, or 3^*4. If w begins with a b or a # each derivation of w can only be of the form 3^*4. In this case we can uniquely deduce the number of applied rules from the length of w. That is w has the unique derivation 3^i4, where $i = \frac{|w|-1}{6}$. If w begins with an a the derivation of w can only be of the form 1^*23^*4 or 1^+4. In this case each derivation of w

has the form 1^+4 if the symbol immediately to the left of the # in w is a b. Then the only possible derivation is 1^i4, where $i = \frac{|w|-1}{6}$. If the first symbol of w is a b and the symbol immediately to the left of the # in w is an a then each derivation for w has the form 1^*23^*4. Note that the productions 1 and 3 generate 6 terminals each, while the remaining two productions generate exactly 3 terminals. Hence, we obtain that each derivation of w has the form 1^i23^j4, where $i + j = \frac{|w|-3}{6}$. Now let u be the longest prefix of w not containing the # symbol. Then each derivation of w has the form 1^i23^j4, where $|u| = 2i + 4j$. Both equations are only satisfied for $i = 2k - m$ and $j = m - k$, where $m := \frac{|u|}{2}$ and $k := \frac{|w|-3}{6}$. Hence, the derivation of w is unique in this case either. Now assume unambiguity has been shown for each word in $L(G)$ which has n bars to the left of the # symbol. Let w be a word in $L(G)$ with $n + 1$ bars on each side of the #-symbol. Now we consider the prefix of w until the leftmost bar and the suffix until the rightmost bar. Then this prefix and suffix can only be produced by a derivation of one of the three forms 1^*23^*4, 1^+4, or 3^*4. The unique derivation of this portion of w can be deduced in the same way as for a word without vertical bars, provided the whole string between the outermost vertical bars is handled as if it is a single #. Thus, we know the beginning of the derivation until the outermost bars. The rest of the derivation produces a word with one bar less on each side of the #. Hence, its derivation is uniquely determined by the inductive hypothesis. Since G is unambiguous and contains only one marker we obtain $G_{\mathrm{Kob}} \in PCFG$. Therefore, according to Theorem 7.54 we obtain $\hat{d}_{G_{\mathrm{Kob}}} \in \Theta(n)$. □

For arbitrary $n, j \in \mathbb{N}$ the grammar $G_{n,j}$ in Theorem 7.54 is a nonterminal bounded grammar with the maximal possible width, which a context-free grammar with n nonterminals and at most j nonterminals on the right-hand side of each production can have. But the price is that each nonterminal belongs to a singleton class with respect to the equivalence relation defined in Definition 2.70. The difficult part of the question, whether the bound of Theorem 7.54 is sharp is whether a minimal linear grammar can have an "ambiguous turn position of infinite degree" without being exponentially ambiguous. This question is solved positive by Lemma 7.55. Now we can combine the grammars $G_{n,j}$ and G_{Kob} to a generalised version of the Kobayashi grammar:

$$\hat{G}_{n,j} := (\{A_1, \ldots, A_n\}, \{a, b\}, P_{n,j}, A_n\}.$$

$$\hat{P}_{n,j} := \begin{cases} \{A_1 \to abA_1baba \mid aA_1a \mid babaA_1ab \mid b\} & \text{for } n = 1 \\ \{A_n \to abA_nbaba \mid aA_na \mid babaA_1ab \mid A_{n-1}\} \cup \hat{P}_{n-1,j} & \text{for } n > 1 \end{cases}$$

It can be shown that for each $n, j \in \mathbb{N}$ the grammar $s(\hat{G}_{n,j})$ is unambiguous. The markers in $s(\hat{G}_{n,j})$ allow to uniquely determine for each occurrence of a nonterminal in a derivation of $s(\hat{G}_{n,j})$ which portion of the word is generated to its left and right, before a descending production is applied. But $\hat{G}_{n,j}$ has the productions of the Kobayashi grammar for each nonterminal. This means that we can apply essentially the reasoning of the previous proof to show that $s(\hat{G}_{n,j})$ is unambiguous. It can also be shown that each marker of $s(\hat{G}_{n,j})$ can occur at a number of positions in a words of the form $(ab)^*a$ which is proportional to its length. Thus, it can be shown that $\hat{d}_{\hat{G}_{n,j}} \in \Theta(n^k)$ where $k := 2\frac{j^n-1}{j-1} - 1$, which means that the bound of Theorem 7.54 is sharp.

Chapter 8

Universal Inherence

At the beginning of [17, 4.7] one can read: "It is easy to exhibit ambiguous context-free grammars [...]. What is not so easy to do is to exhibit a context-free language for which every CFG is ambiguous."

In this chapter we will see that for each cycle-free context-free grammar G there is a context-free language L having the same ambiguity as G. The proof is constructive. Hence, this result turns the "not so easy" part into the easier one. Therefore, to prove that a context-free language with a certain ambiguity function f exists it is sufficient to find a cycle-free context-free grammar with ambiguity function f. Since each ambiguity function for a context-free language is by definition an ambiguity function for some context-free grammar we can formulate the main result of this chapter as follows:

Theorem 8.1 *The set of ambiguity functions for cycle-free context-free grammars and the set of inherent ambiguity functions coincide.*

8.1 Preliminaries

For an arbitrary set of trees $\Delta \subset \Delta_\Gamma$ over an alphabet Γ and a word $w \in \Gamma^*$ we define $\Delta(w) := \{\rho \in \Delta \mid \downarrow(\rho) = w\}$.

The set of mappings from a monoid M to a set S is denoted $S\langle\langle M \rangle\rangle$. An element $s \in S\langle\langle M \rangle\rangle$ is a *formal power series*. For $m \in M$ the value $s(m)$ is called *coefficient* of m. A formal power series can be represented by a formal sum $s := \sum_{m \in M} s(m)m$. For $s \in \mathbb{N}\langle\langle \Sigma^* \rangle\rangle$ we define $\hat{s} : \mathbb{N} \to \mathbb{N}$ by $\hat{s}(n) := \max\{s(w) \mid w \in \Sigma^{\leq n}\}$.

Let $G = (N, \Sigma, P, S)$ be a context-free grammar. The terminals of an embedded tree $\rho \in \mathit{embedded}(\Delta_G) \setminus \Sigma$ can be retrieved from the remaining symbols, i.e., the restriction of the projection $\pi_{P \cup N}$ to $\mathit{embedded}(\Delta_G) \setminus \Sigma$ is injective. Therefore, we define the *parse* of an embedded derivation tree $\rho \in$

embedded$(\Delta_G) \setminus \Sigma$, as a more compact tree representation, by $parse_G(\rho) := \pi_{P \cup N}(\rho)$. The reader familiar with the notion of left parses may note that $parse(\rho)$ and the left parse of ρ coincides for all derivation trees. But in contrast to the left parse, which is only defined for embedded derivation trees of the form $(P \cup \Sigma)^*(N \cup \Sigma)^*$, our parse notion is a unique representation for all embedded derivation trees, but those in Σ. We extend the parse notion in the natural way to sets and observe:

Lemma 8.2 *For each context-free grammar G the sets Δ_G, embedded(Δ_G), $parse_G(\Delta_G)$, and $parse_G($embedded$(\Delta_G))$ are unambiguous context-free languages.*

A very good presentation of Ogden's Lemma can be found in [16, pp185–191]. By Observation 2.57 the result of the pumping tree iteration, implicitly described in the proof of Ogden's Lemma, does not only lead to an iteration of factors on the generated words, but also on the corresponding derivation trees. Translating these observations into our notation we immediately obtain the following version of Ogden's Lemma:

Lemma 8.3 *For each context-free grammar $G = (N, \Sigma, P, S)$ there is an integer $n \in \mathbb{N}$ such that for each $\rho \in \Delta_G$ and any choice of at least n marked positions in ρ there are $\alpha, \beta, \gamma, \delta, \eta \in T_G^*$ and a nonterminal $X \in N$ such that:*

(i) $\rho = \alpha\beta\gamma\delta\eta$.

(ii) *(α and β and γ) or (γ and δ and η) contain at least one marked position.*

(iii) *$\beta\delta$ contains at most n marked positions.*

(iv) $\alpha\beta^i\gamma\delta^i\eta \in \Delta_G$ *and* $\alpha\beta^i X\delta^i\eta$, $\beta^i\gamma\delta^i$, $\beta^i X\delta^i \in$ *embedded*(Δ_G) *for all* $i \in \mathbb{N}$.

A tuple $\vartheta = (|\alpha| + 1, |\alpha\beta|, |\alpha\beta\gamma| + 1, |\alpha\beta\gamma\delta|)$ satisfying the conditions above is called a *pumping phrase* and $\beta\gamma\delta$ the *subtree corresponding to* ϑ. Note that $\rho \in \Delta_G$ implies $\downarrow(\rho) \in L(G)$. Therefore, if we only mark leaves in ρ we obtain Ogden's iteration Lemma for cf languages. The advantage of pumping derivation trees instead of their frontiers is that they have a unique phrase structure even if the generated words are ambiguous. As we will see this additional information can be useful if we generate a sequence of derivation trees by applications of Ogden's iteration Lemma with intermediate shifts of the marked positions.

8.2 The Hiding Theorem

In this section it is shown how the loss of information induced by a length preserving homomorphism can be turned into inherent ambiguity.

For the remainder of this section we define $\Sigma := \{b_1, \ldots, b_k\}$ for some $k \geq 1$, and let Γ and $\{a, \#\}$ be two alphabets such that all three alphabets are pairwise disjoint. Furthermore, let $L \subseteq \Sigma^*$ be an unambiguous context-free language and $h : \Sigma^* \to \Gamma^*$ be a length preserving homomorphism. Each time a $p \in \mathbb{N}$ is used in the sequel we implicitly define $q := p! + p$.

Next we define a system of languages which has an "inherent capacity" to hide information:

Definition 8.4 *For arbitrary $j \in \mathbb{N}$ we write $\langle j \rangle := a^j\#$. For $i \in [1, k]$ we define:*

$$L_i := \{\varepsilon\} \cup \{\langle j_0 \rangle \cdots \langle j_k \rangle \mid j_0, \ldots, j_k \in \mathbb{N} \text{ and } j_0 = j_i\}.$$

All the languages defined in the previous definition are unambiguous.

Definition 8.5 *We define:*

- *The formal power series $s_{h,L}(w) := |h^{-1}(w) \cap L|$.* [1]
- *The substitution $\sigma_h : \Sigma^* \to 2^{(\Gamma \cup \{a,\#\})^*}$ given by $\sigma_h(b_i) := \{h(b_i)\}L_i$ for all $i \in [1, k]$.*
- *The homomorphism $fill_p : \Gamma^* \to (\Gamma \cup \{a, \#\})^*$ defined by $fill_p(X) := X\langle q \rangle^{k+1}$ for all $X \in \Gamma$.*
- *The homomorphism $code_{h,p} : \Sigma^* \to (\Gamma \cup \{a, \#\})^*$ defined by $code_{h,p}(b_i) := h(b_i)\langle p \rangle \langle q \rangle^{i-1} \langle p \rangle \langle q \rangle^{k-i}$ for all $i \in [1, k]$.*

Words in $\sigma_h(L)$ can be broken into blocks *and* subblocks. *A block is an element of $\sigma_h(b_i)$ for some $i \in [1, k]$. They are numbered from left to right beginning with* 1. *The blocks are uniquely determined since they have the form $\Gamma(a^*\#)^{k+1}$ and do not end before the end of the word or the beginning of the next block. A subblock is a word of $a^*\#$ not immediately preceded by an a-symbol. The subblocks are numbered from left to right beginning with* 0.

The main idea of this Chapter is outlined as follows: For each $w \in \Gamma^*$ the coefficient $s_{h,L}(w)$ is the number of words in L which are mapped by h onto w. Thus, it can be seen as the degree of information hiding induced

[1] Formal power series are often denoted as formal sums. In this notation we would write: $s_{h,L} := \sum_{w \in \Gamma^*} |h^{-1}(w) \cap L| \cdot w$.

by h on L. The mapping $code_{h,p}$ is injective since the mapped symbol is coded in the blocks of a- and #-symbols following the image under h. Now for each $u \in \Sigma^*$ both $code_{h,p}(u)$ and $(fill_p \circ h)(u)$ are elements of $\sigma_h(u)$. Thus, $\sigma_h(u)$ contains at the same time words which allow to retrieve u and words which hide all information about u but $h(u)$. Let G be an arbitrary context-free grammar generating $\sigma_h(L)$ and let $u \in L$. We will see that for large enough $p \in \mathbb{N}$ the set Δ_G contains a derivation tree with frontier $(fill_p \circ h)(u)$ obtained by pumping a derivation tree with frontier $code_{h,p}(u)$. Assume $w = h(u_1) = h(u_2)$ for two different words $u_1, u_2 \in L$. Then there are derivation trees ω_1 and ω_2, both having the frontier $fill_p(w)$ obtained by pumping up trees with frontiers $code_{h,p}(u_1)$ and $code_{h,p}(u_2)$, respectively. The main point of Theorem 8.8 is to show that these trees cannot coincide. Thus, the "information hiding" which h induces from the "outside" of L is an inherent feature of $\sigma_h(L)$ "carried out" by the "internal pumping structure" of $\sigma_h(L)$.

As an immediate consequence of the definition we observe:

Observation 8.6

(i) $\forall u \in \Sigma^* : \sigma_h(u) \cap \Gamma^* = \{h(u)\}$ *and*

(ii) $\{h(b_i)\}L_i = \sigma_h(b_i)$ *is an unambiguous context-free language for all* $i \in [1, k]$.

For each $i \in [1, k]$ let $G_i = (N_i, \Gamma \cup \{a, \#\}, P_i, b_i)$ be an unambiguous context-free grammar generating $\sigma_h(b_i)$. Further let $G_L = (N_L, \Sigma, P_L, S)$ be an unambiguous context-free grammar generating L, such that $N_1, \ldots, N_k$, and N_L are pairwise disjoint. Now we define the following grammar:

Definition 8.7

$$G(h, L) := (N_L \cup (\cup_{i\in[1,k]} N_i), \Gamma \cup \{a, \#\}, P_L \cup (\cup_{i\in[1,k]} P_i), S).$$

Note that $L(G(h, L)) = \sigma_h(L)$.

Theorem 8.8 *The substitution* σ_h *has the following properties:*

(i) For all $v \in (\Gamma \cup \{a, \#\})^*$ *we have* $s_{h,L}(\pi_\Gamma(v)) = d_{G(h,L)}(\pi_\Gamma(v)) \geq d_{G(h,L)}(v)$.

(ii) For each context-free grammar G' *such that* $\sigma_h(L) = L(G')$ *there is a constant* $p \in \mathbb{N}$ *such that* $s_{h,L}(w) \leq d_{G'}(fill_p(w))$ *for all* $w \in \Gamma^*$.

The proof of Theorem 8.8 contains all the definitions and lemmas until Theorem 8.17.

Proof of Theorem 8.8 (i): Each derivation tree $\rho \in \Delta_{G(h,L)}$ consists of a tree $\rho' \in \Delta_{G_L} \subseteq \mathit{cut}(\Delta_{G(h,L)})$ appended by subtrees belonging to $\cup_{i\in[1,k]}\Delta_{G_i} \subseteq \mathit{subtree}(\Delta_{G(h,L)})$. In fact, ρ' is uniquely determined by ρ and plays a crucial role in the sequel. Therefore, we define:

Definition 8.9 *The* G_L remainder *of a derivation tree* $\rho \in \Delta_{G(h,L)}$*, denoted by* $\mathit{rem}(\rho)$*, is the uniquely defined derivation tree in* Δ_{G_L} *obtained from* ρ *by truncation of all phrases* $[j, j']$ *for which* $\rho[j, j'] \in \cup_{i\in[1,k]}\Delta_{G_i}$*.*

Observation 8.10 *For all* $\rho \in \Delta_{G(h,L)}$ *the statement* $\downarrow(\rho) \in \sigma_h((\downarrow \circ \mathit{rem})(\rho))$ *is true.*

Proof. The expression $\sigma_h((\downarrow \circ \mathit{rem})(\rho))$ describes the set of words in $\sigma_h(L)$ which are frontiers of those derivation trees in $\Delta_{G(h,L)}$ having the G_L remainder $\mathit{rem}(\rho)$. Obviously, ρ is such a tree. Therefore, $\downarrow(\rho) \in \sigma_h((\downarrow \circ \mathit{rem})(\rho))$. □

Lemma 8.11 *For* $w \in \Gamma^*$ *we have* $(\downarrow \circ \mathit{rem})(\Delta_{G(h,L)}(w)) \subseteq h^{-1}(w) \cap L$*.*

Proof. Let $\rho \in \Delta_{G(h,L)}(w)$ for some $w \in \Gamma^*$. By definition $\mathit{rem}(\rho) \in \Delta_{G_L}$. Thus, $u := (\downarrow \circ \mathit{rem})(\rho) \in L$. It remains to show that $u \in h^{-1}(w)$. By Observation 8.10 we obtain $w = \downarrow(\rho) \in \sigma_h((\downarrow \circ \mathit{rem})(\rho)) = \sigma_h(u)$. Since $w \in \Gamma^*$ we obtain $w \in \sigma_h(u) \cap \Gamma^*$. Then $w = h(u)$ follows by Observation 8.6. This implies $h^{-1}(w) = (h^{-1} \circ h)(u) \ni u$. □

Lemma 8.12 *For arbitrary* $v \in (\Gamma \cup \{a, \#\})^*$ *the restriction of rem to the set* $\Delta_{G(h,L)}(v)$ *is injective.*

Proof. Let $\mathit{rem}(\rho) = \mathit{rem}(\rho')$ for some $\rho, \rho' \in \Delta_{G(h,L)}(v)$ and let $n = |\mathit{rem}(\rho)|$. We can retrieve $\mathit{rem}(\rho)$ from ρ and ρ' by truncation of all those phrases which correspond to subtrees with roots in Σ. Let $\rho_1, \ldots, \rho_n \in \cup_{i\in[1,k]}\Delta_{G_i}$ and $\rho'_1, \ldots, \rho'_n \in \cup_{i\in[1,k]}\Delta_{G_i}$ be these subtrees for ρ and ρ' in a left to right order, respectively. For all $i \in [1, n]$ we observe $\uparrow(\rho_i) = \uparrow(\rho'_i)$. Thus, ρ_i and ρ'_i both must be generated by the same grammar G_{j_i} for some $j_i \in [1, k]$. Since ρ_i and ρ'_i generate the i-th block of v we have $\downarrow(\rho_i) = \downarrow(\rho'_i)$ as well. Then $\rho_i = \rho'_i$ since G_{j_i} is unambiguous. Since $\mathit{rem}(\rho) = \mathit{rem}(\rho')$ and $\rho_i = \rho'_i$ for each $i \in [1, n]$ we obtain $\rho = \rho'$. □

Definition 8.13 *For each $i \in [1,k]$ we define the unique derivation tree $\omega_i \in \Delta_{G_i}$ with the frontier $\downarrow(\omega_i) = h(b_i)$. This tree exists, since $h(b_i) \in \sigma_h(b_i) = L(G_i)$. Moreover, it is unique since G_i is unambiguous. The homomorphism append$: \Delta_{G_L} \to \Delta_{G(h,L)}$ is defined by append$(p) = p$ if $p \in P_L$ and append$(b_i) = \omega_i$ for $i \in [1,k]$.*

Note that for all $\rho \in \Delta_{G_L}$ we have $(\downarrow \circ \mathit{append})(\rho) = (h \circ \downarrow)(\rho)$. Since $\rho = \mathit{rem}(\rho)$ and the G_L remainder is invariant under appending trees we observe that *append* is injective.

Lemma 8.14 *For $w \in \Gamma^*$ we have $h^{-1}(w) \cap L = (\downarrow \circ \mathit{rem})(\Delta_{G(h,L)}(w))$.*

Proof. By Lemma 8.11 we have $h^{-1}(w) \cap L \supseteq (\downarrow \circ \mathit{rem})(\Delta_{G(h,L)}(w))$ for each $w \in \Gamma^*$. Let $u \in h^{-1}(w) \cap L$ for some $w \in \Gamma^*$. Since $u \in L$ there is a $\rho \in \Delta_{G_L}$ with $u = \downarrow(\rho)$. Since all the symbols and productions of G_L are contained in $G(h,L)$ we obtain $\rho \in \mathit{cut}(\Delta_{G(h,L)})$. Let $\rho' := \mathit{append}(\rho) \in \Delta_{G(h,L)}$. Obviously, $\rho = \mathit{rem}(\rho')$. Therefore, $u = \downarrow(\rho) = \downarrow(\mathit{rem}(\rho')) = (\downarrow \circ \mathit{rem})(\rho')$. It remains to show that $\downarrow(\rho') = w$. Since $u \in h^{-1}(w)$ we have $h(u) = w$ and eventually $\downarrow(\rho') = \downarrow(\mathit{append}(\rho)) = (\downarrow \circ \mathit{append})(\rho) = (h \circ \downarrow)(\rho) = h(\downarrow(\rho)) = h(u) = w$. □

Lemma 8.15 *The equation $s_{h,L}(w) = d_{G(h,L)}(w)$ holds for all $w \in \Gamma^*$.*

Proof. Since $s_{h,L}(w) = |h^{-1}(w) \cap L|$ and $d_{G(h,L)}(w) = |\Delta_{G(h,L)}(w)|$ it is sufficient to show that the restriction of $(\downarrow \circ \mathit{rem})$ to $\Delta_{G(h,L)}(w)$ is a bijection onto $h^{-1}(w) \cap L$. By Lemma 8.14 we already know that it is onto $h^{-1}(w) \cap L$. It remains to show that it is injective. Let $(\downarrow \circ \mathit{rem})(\rho) = (\downarrow \circ \mathit{rem})(\rho')$ for some $\rho, \rho' \in \Delta_{G(h,L)}(w)$. Since $\mathit{rem}(\rho), \mathit{rem}(\rho') \in \Delta_{G_L}$ and G_L is unambiguous, $\mathit{rem}(\rho) = \mathit{rem}(\rho')$ follows. By Lemma 8.12 this implies $\rho = \rho'$. Hence, the restriction of $(\downarrow \circ \mathit{rem})$ to $\Delta_{G(h,L)}(w)$ is injective. □

Now we only have to investigate the influence of the projection π_Γ on the ambiguity function to finish the proof of Theorem 8.8 (i).

Lemma 8.16 *The inequality $d_{G(h,L)}(\pi_\Gamma(v)) \geq d_{G(h,L)}(v)$ holds for all $v \in (\Gamma \cup \{a, \#\})^*$.*

Proof. Since $d_{G(h,L)}(v) = |\Delta_{G(h,L)}(v)|$ and $d_{G(h,L)}(\pi_\Gamma(v)) = |\Delta_{G(h,L)}(\pi_\Gamma(v))|$, it suffices to show that the restriction of $(\mathit{append} \circ \mathit{rem})$ to the set $\Delta_{G(h,L)}(v)$ is an injection into the set $\Delta_{G(h,L)}(\pi_\Gamma(v))$. First we show that this mapping is into $\Delta_{G(h,L)}(\pi_\Gamma(v))$. If $\Delta_{G(h,L)}(v) = \emptyset$ this is trivial, otherwise let $\rho \in \Delta_{G(h,L)}(v)$. Since $\mathit{rem}(\rho) \in \Delta_{G_L}$ we obtain $(\downarrow \circ \mathit{rem})(\rho) \in L \subseteq \Sigma^*$. Thus,

we can write $(\downarrow \circ rem)(\rho) = b_{j_1} \cdots b_{j_n}$ for some $j_1, \ldots, j_n \in [1,k]$ and some $n \in \mathbb{N}$. By Observation 8.10 we obtain:

$$\begin{array}{lcll} \pi_\Gamma(\downarrow(\rho)) & \in & \pi_\Gamma(\sigma_h((\downarrow \circ rem)(\rho))) \subseteq \pi_\Gamma(\sigma_h(b_{j_1} \cdots b_{j_n})) & \\ & \subseteq & \pi_\Gamma(h(b_{j_1})\{a,\#\}^* \cdots h(b_{j_n})\{a,\#\}^*) & = \{h(b_{j_1} \cdots b_{j_n})\}. \end{array}$$

By the use of $v = \downarrow(\rho)$ this implies:

$$\begin{array}{lcl} \pi_\Gamma(v) & = & \pi_\Gamma(\downarrow(\rho)) = h(b_{j_1} \cdots b_{j_n}) = h((\downarrow \circ rem)(\rho)) \\ & = & (h \circ \downarrow)(rem(\rho)) = (\downarrow \circ append)(rem(\rho)) = \downarrow((append \circ rem)(\rho)). \end{array}$$

Therefore, $(append \circ rem)(\rho) \in \Delta_{G(h,L)}(\pi_\Gamma(v))$. It remains to show that the restriction of $(append \circ rem)$ to $\Delta_{G(h,L)}(v)$ is injective. This follows by Lemma 8.12 and the observation that *append* is injective. □

Lemma 8.15 and Lemma 8.16 immediately imply Theorem 8.8 (i). □

Proof of Theorem 8.8 (ii): Assume G' is a context-free grammar such that $L(G') = \sigma_h(L)$, p is the maximum of 3 and the pumping constant of G', $q := p! + p$, and $w \in \Gamma^*$. Obviously, $code_{h,p}(h^{-1}(w) \cap L) \subseteq \sigma_h(L)$. In case $h^{-1}(w) \cap L = \emptyset$ the inequality of Theorem 8.8 (ii) is trivially satisfied. Now assume $h^{-1}(w) \cap L \neq \emptyset$. Let $u \in h^{-1}(w) \cap L$, $n := |u|$ and let $j_1, \ldots, j_n \in [1,k]$ be defined such that $u = b_{j_1} \cdots b_{j_n}$. Then we have:

$$code_{h,p}(u) = h(b_{j_1})\langle p\rangle\langle q\rangle^{j_1-1}\langle p\rangle\langle q\rangle^{k-j_1} \cdots h(b_{j_n})\langle p\rangle\langle q\rangle^{j_n-1}\langle p\rangle\langle q\rangle^{k-j_n}.$$

We say that an interval of a derivation tree lies *within* a subblock (block) if the corresponding nodes do not contain any leaf belonging to another subblock (block). We prove by induction that for each $i \in [0,n]$ there is a derivation tree $\rho_i \in \Delta_{G'}$ such that

$$\downarrow(\rho_i) = (fill_p \circ h)(u[1,i]) \cdot code_{h,p}(u[i+1,n])$$

and for each $m \in [1,i]$ the derivation tree ρ_i has a pumping phrase allowing to pump the same number of a-symbols into the 0-th subblock and the j_m-th subblock of block m jointly. For $i = 0$ we only have to show that $code_{h,p}(u) = \downarrow(\rho_0)$ for some $\rho_0 \in \Delta_{G'}$. This follows by $code_{h,p}(u) \in \sigma_h(L) = L(G')$. Assume the statement is true for $i-1$. Then there is a derivation tree $\rho_{i-1} \in \Delta_{G'}$ with the required phrase structure and the sentential form:

$$\downarrow(\rho_{i-1}) = (fill_p \circ h)(u[1,i-1]) \cdot code_{h,p}(u[i,n]).$$

The i-th block of this word has the form

$$code_{h,p}(u[i]) = h(b_{j_i})\langle \underline{p}\rangle\langle q\rangle^{j_i-1}\langle p\rangle\langle q\rangle^{k-j_i}.$$

The 0-th subblock of the i-th block in $\downarrow(\rho_{i-1})$ is underlined to indicate that the leaves of ρ_{i-1} forming the a-symbols of this subblock are marked. According to Ogden's Lemma 8.3 we have $\rho_{i-1} = \alpha\beta\gamma\delta\eta$ for some $\alpha, \beta, \gamma, \delta, \eta \in T^*_{G'}$ such that $\alpha\beta^l\gamma\delta^l\eta \in \Delta_{G'}$ for each $l \in \mathbb{N}$. Moreover, $\beta\delta$ must contain at least one marked position and at least one of the intervals $\tau_\beta := [|\alpha| + 1, |\alpha\beta|]$ and $\tau_\delta := [|\alpha\beta\gamma| + 1, |\alpha\beta\gamma\delta|]$ lies within the 0-th subblock of block i. Let $\tau := \tau_\delta$ if τ_β has this property and $\tau := \tau_\beta$ otherwise.

By the choice of p and q, the insertion of at most p many a-symbols into a subblock $\langle p \rangle$ yields a subblock shorter than $\langle q \rangle$. We will implicitly apply this argument in the sequel.

Assume τ is not within the i-th block, i.e., it is outside or it overlaps with block i and some neighbouring blocks. Let $i' = i + c$ where c is the number of Γ symbols in β if $\tau = \tau_\beta$ and $i' := i$ otherwise. Then block i' of $\downarrow(\alpha\beta^2\gamma\delta^2\eta)$ equals block i of $\downarrow(\alpha\beta\gamma\delta\eta)$, except for a proper insertion of at most p many a-symbols in the 0-th subblock of block i'. Therefore, within block i' the 0-th subblock does not agree with any other subblock.

Now assume τ lies within block i. Then it cannot contain a #-symbol because otherwise the i-th block of $\downarrow(\alpha\beta^2\gamma\delta^2\eta)$ would contain more than $k+1$ subblocks. Hence, each of τ_β and τ_δ lie within one subblock of the i-th block, respectively. We can easily verify that $\downarrow(\alpha\beta^2\gamma\delta^2\eta)$ does not contain more than $2p$ occurrences of a-symbols in the 0-th subblock of block i in this case. This implies that τ_β lies within the 0-th subblock and τ_δ within the j_i-th subblock of block i and $1 \leq |\downarrow(\beta)| = |\downarrow(\delta)| \leq p$. Thus, $l = p! \cdot |\downarrow(\beta)|^{-1} + 1$ is an integer and the derivation tree $\rho_i := \alpha\beta^l\gamma\delta^l\eta$ has the property $\downarrow(\rho_i) = (\mathit{fill}_p \circ h)(u[1, i]) \cdot \mathit{code}_{h,p}(u[i + 1, k])$. Now ρ_i contains a pumping phrase allowing to pump the same number of a-symbols into the 0-th subblock and into the j_i-th subblock of block i jointly. Moreover, the pumping phrases of ρ_i to the left of block i are the same as in ρ_{i-1}, which completes the induction.

Eventually for $i = n$ we obtain a derivation tree ρ_n with the frontier $\downarrow(\rho_n) = (\mathit{fill}_p \circ h)(u) = \mathit{fill}_p(w)$ and the claimed phrase structure starting from an arbitrary word of $\mathit{code}_{h,p}(h^{-1}(w) \cap L)$. It remains to show that two trees obtained in this way beginning with different words in $h^{-1}(w) \cap L$ cannot coincide. Let $u_1, u_2 \in h^{-1}(w) \cap L$ be two different words and let ω_1 and ω_2 be the corresponding derivation trees obtained by the pumping sequence described above. Then ω_1 and ω_2 both generate $\mathit{fill}_p(w)$. Assume $\omega_1 = \omega_2$. Since the two trees are equal we drop the index and define $\omega := \omega_1$. Since h is length preserving we observe $|u_1| = |u_2|$. Therefore, u_1 and u_2 differ in at least one position $i \in [1, |u_1|]$. Then $b_j = u_1[i] \neq u_2[i] = b_{j'}$ for some $j, j' \in [1, k]$. W.l.o.g. we assume $j > j'$. The tree ω contains a pumping phrase ϑ_1 allowing to pump the 0-th subblock and the j-th subblock of block i jointly and it contains a pumping phrase ϑ_2 allowing to pump the 0-th

subblock and the j'-th subblock of block i jointly. Pumping ϑ_1 once within ω we obtain a derivation tree ω' with a word $\beta \in (P \cup \{a\})^*$ inserted to the left of ϑ_2. Since β does only contain leaves labelled with a-symbols ϑ_2 is shifted to the right, but remains within the same subblock as in ω. Thus in ω' it is still possible to pump the 0-th subblock and the j'-th subblock of block i jointly, but now in $\downarrow(\omega')$. This pumping yields a derivation tree ω'' for which the 0-th subblock of block i does no longer agree with any other subblock of block i, which is a contradiction. Hence, $\omega_1 \neq \omega_2$. This implies that $\mathit{fill}_p(w)$ can be generated by at least $|\mathit{code}_{h,p}(h^{-1}(w) \cap L)|$ many different derivation trees. Moreover, since $\mathit{code}_{h,p}$ is injective we finally obtain $d_{G'}(\mathit{fill}_p(w)) \geq |\mathit{code}_{h,p}(h^{-1}(w) \cap L)| = |h^{-1}(w) \cap L| = s_{h,L}(w)$. □

Theorem 8.17 *The context-free language $\sigma_h(L)$ is $\hat{s}_{h,L}$-ambiguous.*

Proof. Recall that $L(G(h,L)) = \sigma_h(L)$. Now Theorem 8.8 (i) implies:

$$\begin{array}{rl} & \max\{d_{G(h,L)}(v) \mid v \in (\Gamma \cup \{a,\#\})^{\leq n}\} \\ \geq & \max\{d_{G(h,L)}(w) \mid w \in \Gamma^{\leq n}\} \\ = & \max\{d_{G(h,L)}(\pi_\Gamma(v)) \mid v \in (\Gamma \cup \{a,\#\})^{\leq n}\} \\ \overset{8.8\ (i)}{\geq} & \max\{d_{G(h,L)}(v) \mid v \in (\Gamma \cup \{a,\#\})^{\leq n}\} \end{array}$$

Hence, all the expressions above are equal and again by Theorem 8.8 (i) we obtain:

$$\begin{array}{rcl} \hat{s}_{h,L}(n) & = & \max\{s_{h,L}(w) \mid w \in \Gamma^{\leq n}\} \overset{8.8\ (i)}{=} \max\{d_{G(h,L)}(w) \mid w \in \Gamma^{\leq n}\} \\ & = & \max\{d_{G(h,L)}(v) \mid v \in (\Gamma \cup \{a,\#\})^{\leq n}\} = \hat{d}_{G(h,L)}(n) \end{array}$$

Thus, $G(h,L)$ is appropriate to satisfy property (i) of Definition 2.96. By Theorem 8.8 (ii) we obtain that for each context-free grammar G' such that $L(G') = \sigma_h(L)$ there is a $p \in \mathbb{N}$ such that for all words $w \in \Gamma^*$ we have $s_{h,L}(w) \leq d_{G'}(\mathit{fill}_p(w))$. This implies $\hat{s}_{h,L}(|w|) \leq \hat{d}_{G'}(|\mathit{fill}_p(w)|) = \hat{d}_{G'}(c \cdot |w|)$ where $c = 1 + (k+1)(p! + p + 1)$. Thus, $\sigma_h(L)$ and $\hat{s}_{h,L}$ also satisfies property (ii) of Definition 2.96. Hence, $\sigma_h(L)$ is $\hat{s}_{h,L}$-ambiguous. □

8.3 Applications

8.3.1 Census Functions

Definition 8.18 *Let $L \subseteq \Sigma^*$ be a formal language. The* census function *$\gamma_L : \mathbb{N} \to \mathbb{N}$ is defined by $\gamma_L(n) := |\Sigma^n \cap L|$, and the function $\hat{\gamma}_L : \mathbb{N} \to \mathbb{N}$ is defined by $\hat{\gamma}_L(n) := \max\{\gamma_L(i) \mid i \leq n\}$. The homomorphism hide : $\Sigma^* \to \{\$\}$ is defined by hide$(X) := \$$ for all $X \in \Sigma$.*

Theorem 8.19 *Let $L \subseteq \Sigma^*$ be an unambiguous context-free language. Then*

$$\sigma_{hide}(L) \text{ is } \hat{\gamma}_L\text{-ambiguous.}$$

Proof. By Theorem 8.17 the language $\sigma_{hide}(L)$ is $\hat{s}_{hide,L}$-ambiguous. We obtain $\hat{s}_{hide,L}(n) = \max\{s_{hide,L}(w) \mid w \in \Sigma^{\leq n}\} = \max\{|hide^{-1}(w) \cap L| \mid w \in \Sigma^{\leq n}\} = \max\left\{|\Sigma^{|w|} \cap L| \,\middle|\, w \in \Sigma^{\leq n}\right\} = \max\left\{|\Sigma^j \cap L| \,\middle|\, j \leq n\right\} = \max\left\{\gamma_L(j) \,\middle|\, j \leq n\right\} = \hat{\gamma}_L(n)$. □

Corollary 8.20 *There is an unambiguous context-free language L such that L^+ is exponentially ambiguous and L^k is $\Theta(n^{k-1})$-ambiguous for each $k \in \mathbb{N} \setminus \{0\}$.*

Proof. Let $\{a, b\}$ be an alphabet. We observe that $\hat{\gamma}_{(a+b)^*b}(n) = \lfloor 2^{n-1} \rfloor$ and $\hat{\gamma}_{(a^*b)^k}(n) = \binom{n-1}{k-1}$ for each $k \in \mathbb{N} \setminus \{0\}$. Using Theorem 8.19 we obtain that $\sigma_{hide}((a+b)^*b)$ is $\lfloor 2^{n-1} \rfloor$-ambiguous and $\sigma_{hide}((a^*b)^k)$ is $\binom{n-1}{k-1}$-ambiguous. Thus, $\sigma_{hide}((a^*b)^1)$ is unambiguous. Finally, since σ_{hide} is a homomorphism we immediately get $\sigma_{hide}((a+b)^*b) = \sigma_{hide}((a^*b)^+) = (\sigma_{hide}(a^*b))^+$ and $\sigma_{hide}((a^*b)^k) = (\sigma_{hide}(a^*b))^k$. Thus, $L := \sigma_{hide}(a^*b)$ is a language with the required properties. □

8.3.2 Grammars in Greibach Normal Form

In this section we show that the ambiguity function of each context-free grammar is inherent for some context-free language.

For context-free grammars in Greibach normal form the parse of each derivation tree has the same length as its frontier. Moreover, the i-th symbol of the frontier is uniquely determined by the i-th symbol of the parse. This implies the following lemma:

Lemma 8.21 *Let $G = (N, \Sigma, P, S)$ be a context-free grammar in Greibach normal form and $h_G : P^* \to \Sigma^*$ the length preserving homomorphism defined by $h_G(p) := X_p$ for each $p \in P$ where X_p is the terminal at the beginning of p's right-hand side. Then $\downarrow(\rho) = h_G(parse(\rho))$ for all $\rho \in \Delta_G$.*

Theorem 8.22 *Let $G = (N, \Sigma, P, S)$ be a context-free grammar in Greibach normal form, and h_G defined as in Lemma 8.21. Then the context-free language $\sigma_{h_G}(parse_G(\Delta_G))$ is $\hat{d}_G$-ambiguous.*

Proof. Let $L := parse_G(\Delta_G)$ and $h := h_G$.

$$\begin{aligned}
\hat{s}_{h,L}(n) &= \max\left\{|h^{-1}(w) \cap L| \,\middle|\, w \in \Sigma^{\leq n}\right\} \\
&= \max\left\{\left|\left\{\rho \in \Delta_G \,\middle|\, \downarrow(\rho) = w\right\}\right| \,\middle|\, w \in \Sigma^{\leq n}\right\} \\
&= \max\left\{d_G(w) \,\middle|\, w \in \Sigma^{\leq n}\right\} \qquad = \quad \hat{d}_G(n).
\end{aligned}$$

By Lemma 8.2 the context-free language $parse_G(\Delta_G)$ is unambiguous. Moreover, h_G is length preserving. Thus, the claim follows by Theorem 8.17 $\square$

8.3.3 Cycle-Free Context-Free Grammars

Here we complete the proof of Theorem 8.1. stating that the set of ambiguity functions for cycle-free context-free grammars and the set of inherent ambiguity functions coincide. By definition each inherent ambiguity function is the ambiguity function of a context-free grammar. It is easily seen that there is also a cycle-free context-free grammar with this property (Observation 8.23). To show that each ambiguity function of a cycle-free context-free grammar G is inherent for some context-free language is much more tricky. The essential idea is to transform G in an ambiguity preserving way into Greibach normal form and than apply Theorem 8.22. Unfortunately, we cannot preserve the ambiguity in a strict sense, since Greibach normal form grammars cannot generate the empty word and they cannot generate words of length one ambiguously. Thus, we will first transform G in a Greibach normal form grammar with almost the same ambiguity function. Than we will handle the differences between the ambiguity functions.

Observation 8.23 *Each inherent ambiguity function is the ambiguity function of a cycle-free context-free grammar.*

Proof. The constant function $f_0 : \mathbb{N} \to \mathbb{N}$ defined by $f_0(n) = 0$ for each $n \in \mathbb{N}$ is inherent for the language $\emptyset \subseteq \Sigma^*$. For some symbol $S \notin \Sigma$ we define the cycle-free context-free grammar $G = (\{S\}, \Sigma, \emptyset, S)$. Now we observe that $\hat{d}_G = f_0$. (Note that there is no reduced context-free grammar in this case, since each context-free grammar generating $\emptyset$ has a useless start symbol.)

Let $f : \mathbb{N} \to \mathbb{N}$ be an arbitrary inherent ambiguity function, such that $f \neq f_0$. Then there is an f-ambiguous context-free language $L \neq \emptyset$ and a context-free grammar G such that $\hat{d}_G = f$ and $L = L(G)$. Since $L(G) \neq \emptyset$ at least the start symbol of G is useful. Moreover, symbols which cannot occur in any derivation tree do not contribute to the ambiguity. Thus, we can assume without loss of generality that G is reduced. Assume G is cyclic, then there is a word u which has infinitely many derivation trees. Then $\hat{d}_G(|u|) = \omega \neq f(|u|)$ contradicting $\hat{d}_G = f$. Hence G is cycle-free. $\square$

Note that any context-free language can be generated by a cycle-free context-free grammar and by [16, Theorem 12.2.1] the ambiguity function of each cycle-free context-free grammar G grows at most exponentially ($\hat{d}_G \in \mathcal{O}^T(2^n)$). In particular, $\hat{d}_G(n) \in \mathbb{N}$ for each $n \in \mathbb{N}$. Hence, the fact

that Definition 2.96 and Observation 8.23 restricted the range of f to $\mathbb{N}$ instead of allowing $\mathbb{N} \cup \{\omega\}$ is not crucial for the validity of Observation 8.23.

[22, Corollary 5.12.] claims that any cycle-free context-free grammar $G = (N, \Sigma, P, S)$ can be transformed into a grammar G' in Greibach normal form such that $d_G(w) = d_{G'}(w)$ for each $w \in \Sigma^+$. Unfortunately, this is not necessarily true for words of length one. A grammar in Greibach normal form does neither have ε-productions nor chain productions. Without them it is impossible to generate a word of length one ambiguously. But by the use of a chain production it is easy to obtain an ε-free cycle-free context-free grammar which generates a word of length one ambiguously. For instance by a context-free grammar defined by the following production set:

$$S \to A \mid a, \quad A \to a$$

Fortunately, the author of [22] just drew a wrong conclusion from the correct statements [22, Theorems 3.9, 5.4, 5.10] and [29, Lemma IV.2.5]. In particular, he has overseen that [29, Lemma IV.2.5] introduces terms consisting of a single variable. They correspond to chain productions not allowed in a Greibach normal form.

Thus the author of [22] actually proved the following statement:

Lemma 8.24 *For each cycle-free context-free grammar $G = (N, \Sigma, P, S)$ there is a context-free grammar $G' = (N_1 \cup N_2, \Sigma, P', S)$ where $N_1 \cap N_2 = \emptyset$ and*

$$P' \subseteq (N_1 \times (\Sigma(N_1 \cup N_2)^* \cup N_2)) \cup (N_2 \times \Sigma).$$

Moreover, $L(G') = L(G) \setminus \{\varepsilon\}$ and for each $w \in \Sigma^+$ the equation $d_G(w) = d_{G'}(w)$ is satisfied.

Proof. By [22, Theorem 3.9] the ambiguity power series d_G is the first component of the solution of the algebraic system $\mathcal{A}_G$ corresponding to G (see [22] for the definition of algebraic systems corresponding to context-free grammars.) We apply the construction of [22, Theorem 5.10], resulting in an equivalent algebraic system only containing ε-terms and terms in Greibach normal form, the latter terms consist of a single symbol followed by a sequence of variables. (In addition the length of the terms is bounded by three.) The construction of [22, Theorem 5.4], eliminating ε-terms, is than applied to the resulting algebraic system yielding an algebraic system $\mathcal{A}'_G$, whose terms are in Greibach normal form. Moreover the solution of the first component of $\mathcal{A}'_G$ is d_G^+, where $d_G^+(\varepsilon) := 0$ and $d_G^+(w) := d_G(w)$ for all $w \in \Sigma^+$. Observe that the elimination of ε-terms, does not introduce terms not in Greibach normal form. The construction of [29, Lemma IV.2.5] is

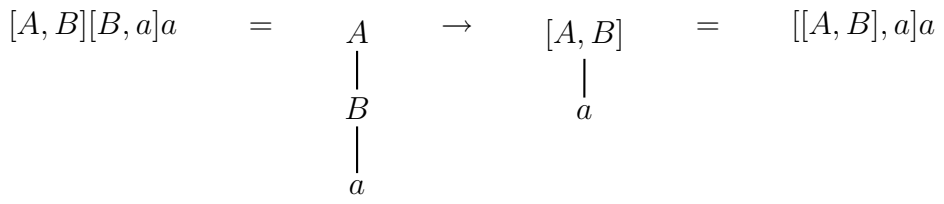


Figure 8.1: chain production $[A, B]$ compressed into nonterminal $[A, B]$

meant to avoid coefficients larger than one by introducing several copies of variables. Essentially it does not change the form of the terms, but it introduces terms consisting of a single variable. Originally it transforms a systems whose terms are in Chomsky normal form (see [22] for a definition). But a completely analogous transformation for an algebraic system with terms in Greibach normal form yields an algebraic system ${\mathcal{A}''}_G$ corresponding to G' □

Lemma 8.25 *Let $G = (N_1 \cup N_2, \Sigma, P, S)$ be a cycle-free context-free grammar, where $N_1 \cap N_2 = \emptyset$ and*

$$P \subseteq (N_1 \times (\Sigma(N_1 \cup N_2)^* \cup N_2)) \cup (N_2 \times \Sigma).$$

Then there is a context-free grammar $G' = (N', \Sigma, P', S)$ in Greibach normal form, such that $L(G') = L(G) \setminus \{\varepsilon\}$ and $d_G(w) = d_{G'}(w)$ for each $w \in \Sigma^{\geq 2}$.

Proof. It suffices to construct G' such that the tree manipulation described in Figure 8.1 gives rise to a bijection between the embedded trees of G and G'. Thus chain productions of G are nonterminals of G'. Whenever a nonterminal A is generated on the right-hand side of a production $p \in P$ and $[A, B]$ is a chain production then there is a copy of p generating the nonterminal $[A, B]$ instead of A. Let $P'' := (N_1 \times N_2) \cap P$, i.e., P'' is the set of chain productions in P. Let $G' := (N', \Sigma, P', S)$, where $N' := N_1 \cup P''$. The mapping $h : N'^* \to N_1^*$ is the homomorphism defined by:

$$h(X) = \begin{cases} X & \text{if } X \in N_1 \\ \ell(X) & \text{if } X \in P'' \end{cases}$$

Finally:

$$\begin{aligned} P' \;:=\; & \{[[A, B], a] \mid [A, B] \in P'' \text{ and } [B, a] \in P\} \\ \cup\; & \{[A, \alpha] \in N_1 \times \Sigma N'^* \mid [A, h(\alpha)] \in P\}. \end{aligned}$$

The first set in the definition of P' compresses chain productions, the second define when the new nonterminals occur on right-hand sides of productions. It is easily seen that the construction has the desired properties. □

Note that the result can easily be strengthened to allow at most two nonterminals on a right-hand side, but we do not need that.

Lemma 8.24 and Lemma 8.25 immediately imply:

Lemma 8.26 *For each cycle-free context-free grammar $G = (N, \Sigma, P, S)$ there is a context-free grammar $G' = (N', \Sigma, P', S)$ in Greibach normal form, such that $L(G') = L(G) \setminus \{\varepsilon\}$ and $d_G(w) = d_{G'}(w)$ for each $w \in \Sigma^{\geq 2}$.*

Lemma 8.27 *Let $G = (N, \Sigma, P, S)$ be a cycle-free context-free grammar. Then there is a cycle-free context-free grammar G' such that*

$$d_{G'}(u) = \begin{cases} d_G(u) & \text{if } u \in \Sigma^{\geq 2} \\ 0 & \text{if } u \in \Sigma^{\leq 1} \end{cases}$$

Proof. Let $\mathcal{S} := \{\alpha \in (N \cup \Sigma)^* \mid \exists v \in \Sigma^{\leq 1} : S \Rightarrow^*_{\mathrm{lm},G} \alpha \Rightarrow^*_{\mathrm{lm},G} v\}$, i.e., $\mathcal{S}$ is the set of left-sentential forms[2] of G which can derive the empty word or a single terminal. Since G is cycle-free each word has only a finite number of (leftmost) derivations. Moreover, each derivation has finite length and $\Sigma^{\leq 1}$ is finite. Hence $\mathcal{S}$ is finite. In case $\mathcal{S} = \emptyset$ the claim is trivially satisfied by setting $G' := G$. Now we consider the case $\mathcal{S} \neq \emptyset$. Let N_1 be a disjoint copy of $\mathcal{S}$, i.e., $|N_1| = |\mathcal{S}|$ and $N_1 \cap \mathcal{S} = \emptyset$. Moreover, we associate a unique element $\bar{\alpha} \in N_1$ with each element $\alpha \in \mathcal{S}$. Since $\mathcal{S}$ is non-empty we have $S \in \mathcal{S}$. Hence $\bar{S} \in N_1$. Now we construct the cycle-free context-free grammar $G' = N, \Sigma, P \cup P', \bar{S})$ where

$$\begin{aligned} P' \quad := \quad & \{(\bar{\alpha}, \bar{\beta}) \in N_1^2 \mid \alpha \Rightarrow^*_{\mathrm{lm},G} \beta\} \\ \cup \quad & \{(\bar{\alpha}, \beta) \in N_1 \times ((N \cup \Sigma)^* \setminus \mathcal{S}) \mid \alpha \Rightarrow^*_{\mathrm{lm},G} \beta\} \end{aligned}$$

There is an obvious bijection between left-sentential forms of G and G'. It is the mapping $\varphi : (N \cup \Sigma)^* \to N_1 \cup (N \cup \Sigma)^*$ defined by:

$$\varphi(\alpha) := \begin{cases} \bar{\alpha} & \text{if } \alpha \in \mathcal{S} \\ \alpha & \text{otherwise.} \end{cases}$$

A straightforward induction shows that:

$$\forall \alpha, \beta \in (N \cup \Sigma)^* : ((\alpha \Rightarrow^*_{\mathrm{lm},G} \beta) \Leftrightarrow (\varphi(\alpha) \Rightarrow^*_{\mathrm{lm},G'} \varphi(\beta))).$$

[2] A sentential form is a *left-sentential form* if it is generated by a leftmost derivation.

Thus the number of cuts of derivation trees generating a left-sentential form α by using G equals the number of cuts of derivation trees generating $\varphi(\alpha)$ by using G'. Note that $\bar{v}$ is a useless nonterminal for each $v \in \Sigma^{\leq 1}$ while $\varphi(u) \in L(G')$ for each $u \in \Sigma^{\geq 2}$. Hence $d_{G'}$ has the claimed properties. □

Lemma 8.28 *Each ambiguity function of a cycle-free context-free grammar is inherent for some context-free language.*

Proof. Let G_1 be a cycle-free context-free grammar. By Lemma 8.26 there is a grammar G_2 in Greibach normal form such that $L(G_2) = L(G_1) \setminus \{\varepsilon\}$ and $d_{G_2}(w) = d_{G_1}(w)$ for each $w \in \Sigma^{\geq 2}$. If necessary, we add some nonterminals and productions such that for some $u \in \Sigma^2$ the resulting Greibach normal form grammar G_3 has the properties:

(i) $\hat{d}_{G_2}(2) = d_{G_3}(u)$

(ii) $\forall v \in \Sigma^* \setminus \{u\} : d_{G_3}(v) = d_{G_2}(v)$.

Then $L(G_2) \subseteq L(G_3) \subseteq L(G_2) \cup \{u\}$. This modification may change the generated language but it guarantees that $d_{G_3}(n) = d_{G_1}(n)$ for all $n \geq 2$. Moreover $d_{G_3}(n) \leq d_{G_1}(n)$ for all $n \in \mathbb{N}$. By Theorem 8.22 there is an inherently $\hat{d}_{G_3}$-ambiguous context-free language L. Thus there is a cycle-free context-free grammar G_4 such that $L = L(G_4)$ and $\hat{d}_{G_4} = \hat{d}_{G_3}$. If $\hat{d}_{G_1} \neq \hat{d}_{G_4}$ then we add some nonterminals, chain productions and ε-productions only changing the ambiguity of words of length at most one. We do this such that we get a cycle-free context-free grammar G_5 with the property $\hat{d}_{G_1} = \hat{d}_{G_5}$. Moreover, $d_{G_5}(w) = d_{G_4}(w)$ for all $w \in \Sigma^{\geq 2}$. This implies $L(G_5) \setminus \Sigma^{\leq 1} = L \setminus \Sigma^{\leq 1}$. Now we show that $L(G_5)$ is inherently $\hat{d}_{G_1}$-ambiguous. We have already shown that $L(G_5)$ is generated by a context-free grammar with the ambiguity-function $\hat{d}_{G_1}$, namely G_5. It remains to show that for each context-free grammar G' generating $L(G_5)$ there is a constant $c \in \mathbb{N}$ such that for each $n \in \mathbb{N} \setminus \{0\}$ the relation $\hat{d}_{G'}(cn) \geq \hat{d}_{G_1}(n)$. Now assume that G' is an arbitrary context-free grammar generating $L(G_5)$. By Lemma 8.27 there is a cycle-free context-free grammar G'' such that:

$$d_{G''}(u) = \begin{cases} d_{G'}(u) & \text{if } u \in \Sigma^{\geq 2} \\ 0 & \text{if } u \in \Sigma^{\leq 1} \end{cases}$$

We have:

$$L(G'') = L(G') \setminus \Sigma^{\leq 1} = L(G_5) \setminus \Sigma^{\leq 1} = L \setminus \Sigma^{\leq 1}.$$

By adding some nonterminals, chain-productions, and ε-productions be obtain a grammar G''' such that

$$d_{G'''}(u) = \begin{cases} d_{G'}(u) & \text{if } u \in \Sigma^{\geq 2} \\ 1 & \text{if } u \in \Sigma^{\leq 1} \text{ and } u \in L \\ 0 & \text{if } u \in \Sigma^{\leq 1} \text{ and } u \notin L \end{cases}$$

Now we have $L(G''') = L$. Since L is inherently $\hat{d}_{G_3}$-ambiguous there is a constant $c \in \mathbb{N}$ such that $\hat{d}_{G'''}(cn) \geq \hat{d}_{G_3}(n)$ for each $n \in \mathbb{N} \setminus \{0\}$. For a $c \in \mathbb{N}$ with this property and each $n \in \mathbb{N} \setminus \{0\}$ we have

$$\hat{d}_{G'}(2cn) = \hat{d}_{G'''}(2cn) \geq \hat{d}_{G_3}(2n) = \hat{d}_{G_1}(2n) \geq \hat{d}_{G_1}(n)$$

Hence for $c' := 2c$ we have $\hat{d}_{G'}(c'n) \geq \hat{d}_{G_1}(n)$ for each $n \in \mathbb{N} \setminus \{0\}$. Thus, $L(G_5)$ is inherently $\hat{d}_{G_1}$-ambiguous. □

Finally, Observation 8.23 and Lemma 8.28 imply Theorem 8.1.

8.3.4 Sublinear Ambiguity Functions

In the last section we have completed the proof of Theorem 8.1. Thus, the set of ambiguity functions for cycle-free context-free grammars and the set of inherent ambiguity functions coincide. This result allows to transfer our results on sublinear ambiguity functions in Theorem 6.18 to context-free languages. Hence, we finally obtain

Theorem 8.29 *If $f : \mathbb{N} \to \mathbb{N}$ is a computable divergent total non-decreasing function then there is a context-free language L such that L has a divergent inherent ambiguity function $\hat{d}_L$ with the property $\hat{d}_L(n) \leq f(n)$ for all $n \in \mathbb{N}$.*

Theorem 8.1 also allows to show a result on the product of two inherent ambiguity functions:

Theorem 8.30 *Let $f, g : \mathbb{N} \to \mathbb{N}$ be two inherent ambiguity functions. Then there is an ambiguity function in $\Theta^T((f \cdot g)(n))$.*

Proof. If $f, g : \mathbb{N} \to \mathbb{N}$ be two inherent ambiguity functions then by definition there are two reduced context-free grammars $G_f = (N, \Sigma, P, S)$ and G_g which are f- and g-ambiguous, respectively. We also require that G_f does not generate the empty word. Since both grammars do not generate a single word with infinite ambiguity they are cycle-free. Now we define the context-free grammar $\bar{G}_f := (N \dot{\cup} \bar{N}, \Sigma \dot{\cup} \bar{\Sigma}, P \cup \bar{P}, \bar{S})$, where

$$\bar{P} := \{[\bar{A} \to \alpha \bar{X}] \mid A \in N, \alpha \in (N \cup \Sigma)^*, \bar{X} \in \bar{N} \cup \bar{\Sigma}, [A, \alpha X] \in P\}.$$

This grammar has essentially the same set of derivation trees except for the fact that for each $\rho \in cut(\Delta_{\bar{G}_f})$ the rightmost leaf is marked. Hence, $\hat{d}_{\bar{G}_f} = \hat{d}_{G_f} = f$. Due to the marking of the rightmost symbol $L(\bar{G}_f)L(G_g)$ is an unambiguous concatenation. Thus, according to Corollary 4.4 there is a reduced context-free grammar G with an ambiguity function $\hat{d}_G \in \Theta^T((f \cdot g)(n))$. Since G generates no word with infinitely many derivation trees it is cycle-free. Finally, by Theorem 8.1 this implies that $\hat{d}_G$ is an inherent ambiguity function. □

Chapter 9

Conclusion

For a summary of the results the reader is referred to the Introduction. Here additional remarks on some results, suggestions for further research, and open problems are presented.

9.1 Remarks on the Results

We have seen that a cycle-free context-free grammar G is exponentially ambiguous if its set of pumping trees is ambiguous.[1] Moreover, we have defined a context-free grammar $s(G)$ obtained from G by adding some markers into the right-hand sides of bounded productions in a well defined way. If the set of G's pumping trees is unambiguous then $s(G)$ is unambiguous too. This leads to a polynomial upper bound for the ambiguity of G, whose polynomial degree is bounded by the maximal number of markers k contained in a word of $L(s(G))$. We can compute k efficiently with respect to the size of G. Then we know that G is either $\mathcal{O}(n^k)$-ambiguous or $\Theta^T(2^n)$-ambiguous. However, the question whether G is exponentially ambiguous or not is undecidable. The gap between exponential ambiguity and polynomially bounded ambiguity has been already discovered as a part of the authors Diplomarbeit. But here k is improved in many cases substantially. Moreover, the proof is simpler and algebraically more satisfying. Furthermore, $s(G)$ and its relation to G is new in this thesis. This relation has been used to prove that

[1] It is easily seen that cycle-free context-free grammars are at most exponentially ambiguous. Cyclic context-free grammars may exceed this ambiguity by generating some words with infinite ambiguity. This happens if and only if a cycle exists for some useful nonterminal. It is easy to detect this situation and to transform such a grammar into an equivalent cycle-free grammar. Hence, to have infinite ambiguity for a single word is not an inherent feature of any context-free language. Therefore, the investigation of cyclic context-free grammars is not very interesting.

bounded marker languages *BMCFL* and the class of languages with polynomially bounded ambiguity *PCFL* coincide. In other words the class *PCFL* is the closure of the class of unambiguous context-free languages *UCFL* under bounded contraction. While bounded contractions already suffices to generate *PCFL* from the class of unambiguous languages even the formally far stronger operation of bounded substitution does not suffice to leave *PCFL*, i.e., *PCFL* is closed under bounded substitution.

In contrast to the huge gap between $ECFL$ and *PCFL* there is no such gap between the classes of languages with a bounded degree of ambiguity *FCFL* and with infinite ambiguity. We have shown that for each computable divergent total non-decreasing function f there is a context-free grammars G with infinite ambiguity whose ambiguity function is bounded by f, i.e., $\forall n \in \mathbb{N} : \hat{d}_G(n) \leq f(n)$. This allows infinite sequences of context-free grammars with strictly decreasing ambiguity. (To make this statement precise we define that an infinite sequence of context-free grammars $G_0, G_1, \ldots$ has strictly decreasing ambiguity if $\hat{d}_{G_i} \in o^T(\hat{d}_{G_{i+1}})$ for each $i \in \mathbb{N}$.) This leads immediately to the question whether there are such strictly decreasing sequences of context-free grammars such that all the grammars in the sequence generate the same language.[2] If this is possible it would mean that there are context-free languages which does not have a corresponding inherent ambiguity function.

Finally, we examined the relationship between ambiguity functions of context-free grammars and the ambiguity functions of context-free languages. In [17, 4.7] one can read:

"It is easy to exhibit ambiguous context-free grammars [...]. What is not so easy to do is to exhibit a context-free language for which every CFG is ambiguous."[3]

One can generalise this statement to ambiguity functions generated by arbitrary context-free grammar. Then the "not so easy" part is the question whether the ambiguity function of each cycle-free context-free grammar is inherent for some context-free language. We have answered this question positive. (If we allow cyclic context-free grammars the answer is trivially no.) Thus, the set of ambiguity functions for context-free languages and the set of ambiguity functions for cycle-free context-free grammars coincide. Moreover, the proof is constructive, i.e., for a given cycle-free context-free grammar G an $\hat{d}_G$-ambiguous context-free language is explicitly given. This turns the "not so easy" part of the statement above into the easy one: Theorem 8.22

[2] If we replace the o^T notation in the definition of strictly decreasing ambiguity by the o notation then this is possible for exponentially ambiguous languages [24]. This is due to the fact that $\Theta^T(2^n) \supset \Theta(2^{kn})$ for each $k \in \mathbb{N}$, while $\Theta(2^{k_1 n}) \neq \Theta(2^{k_2 n})$ for $k_1 \neq k_2$.

[3] The notion CFG means context-free grammar here.

directly provides an $\hat{d}_G$-ambiguous language for each cycle-free context-free grammar G. If one is interested in the question which ambiguity functions are possible for context-free grammars this is a useful result. All the ambiguity functions of context-free grammars which we can create by the constructions described in Chapter 6 are inherent for some context-free language. (Note that the constructed grammars are cycle-free) Thus, for each computable divergent total non-decreasing function f there is a context-free language L with infinite ambiguity whose ambiguity function is in $\mathcal{O}^T(f)$, Moreover, Theorem 8.22 provides a general construction principal. For instance the right linear grammars over a single letter alphabet in Chapter 3 are sufficient to witness the existence of context-free languages of any finite degree, any polynomial degree, and of exponential degree. So far each of the ambiguity types mentioned above has been considered separately [23, 24]. But Theorem 8.22 covers the common techniques used in all these proofs.

In fact, Theorem 8.22 is deduced from Theorem 8.8, which is more technical, but somewhat stronger. Theorem 8.8 does not deal with ambiguity functions but with ambiguity series. Intuitively it states that the loss of information induced by an arbitrary length preserving homomorphism can be turned into inherent ambiguity. One consequence is the following: Let G_1 and G_2 be cycle-free context-free grammars. Let s be the Chauchy product of d_{G_1} and d_{G_2} defined by $s(w) := \sum_{w=uv} d_{G_1}(u)d_{G_2}(v)$ for each word w. Then there is an $\hat{d}_{G_1}$-ambiguous context-free language L_1 and an $\hat{d}_{G_2}$-ambiguous context-free language L_2 such that L_1L_2 is $\hat{s}$ ambiguous where $\hat{s} : \mathbb{N} \to \mathbb{N}$ is defined by $\hat{s}(n) := \max\{s(w) \mid w \in \Sigma^{\leq n}\}$.

It is hard to find a non-trivial characterisation for the class of languages with bounded ambiguity. There is no non-trivial criterion known, which separates the class of context-free grammars with finite ambiguity (*FCFL*) from the class of context-free grammars with infinite ambiguity. One reason might be that there are "almost constantly ambiguous" context-free grammars with infinite ambiguity (see Chapter 6) Maybe "almost constantly ambiguous" context-free grammars are too similar to context-free grammars with a constant degree of ambiguity to allow simple non-trivial separations. In fact, we have seen that "almost constantly ambiguous" context-free grammars can be parsed in "almost quadratic time" (Theorem 5.9). In contrast to that we have a huge gap between *PCFG* and *ECFG*. We know two non trivial criteria separating these classes (ambiguity of pumping trees and the ambiguity of $s(G)$.) Moreover, *PCFL* is closed under bounded contractions. In addition, we know that *PCFL* is the closure of *UCFL* under bounded contractions. This relationship between *UCFL* and *PCFL* also leads to the result that the languages in *PCFL* can be parsed in logarithmic time on a CREW-PRAM.

All this indicates that the distinction between polynomial and exponential ambiguity might be more natural than the one between a finite degree of ambiguity and an infinite degree.

9.2 Some Suggestions for further Research

So far no nontrivial characterisation of the set of ambiguity functions is known. Due to Theorem 8.1, it is not necessary to consider this question for cycle-free context-free grammars and context-free languages separately. We have also gathered a lot of knowledge about the nature of inherent ambiguity functions:

(i) There is no ambiguity function in $\omega(2^{\mathcal{O}(n)})$ (trivial).

(ii) There is no ambiguity function in $o(2^{\Omega(n)}) \cap \omega(n^{\mathcal{O}(1)})$ (see 7.37).

(iii) According to Theorem 6.18 there are ambiguity functions, which grow as slowly as any computable function.

(iv) Let $f, g : \mathbb{N} \to \mathbb{N}$ be two inherent ambiguity functions. Then there is an inherent ambiguity function in $\Theta^T((f \cdot g)(n))$ (see 8.30).

But all these results do not determine the set of inherent ambiguity functions completely. Theorem 6.18 only claims that substantially below a divergent inherent ambiguity function we can always find another one. It does not tell anything about the question whether there are further gaps, like the one betwen exponential and polynomially bounded ambiguity.

In the proof of Theorem 6.18 for each Turing machine M a context-free grammar G is constructed, which generates essentially a superset of the set of valid computation. In addition, the ambigity function is dominated by the ambiguity of the valid computations. The ambiguity of a valid computation is the number of occurrences of a certain state. By the choice of the Turing machine we can control this number quite well. This indicates that a fine tuning of sublogarithmic ambiguity is possible. This could lead to good density results for slowly growing ambiguity functions.

Unfortunately, the estimation of the length of valid computations is a mixture of time and space complexity. The length of a computation is a series over the time where each addend is the space used at the corresponding time. On the other hand it is not necessary to stick to Turing machines as a means of computation. Instead of the transition relation of a Turing machine one can use unambiguous context-free relations to describe the transition of two configurations.

Another field of research is the examination of subsets of languages responsible for the ambiguity. We cannot expect to determine such a subset uniquely. Note that there is no single word w in a context-free language L, which is generated ambiguously by each context-free grammar G generating L. Instead we allow some amount of fuzzyness:

Definition 9.1 *Let L be a context-free language and $k \in \mathbb{N}$. A subset $L' \subseteq L$ is a* complete at least k-ambiguous subset *of L if the following conditions are satisfied:*

(i) Each context-free grammar G with $L = L(G)$ generates all but a finite subset of words in L' with at least k derivation trees.

(ii) There is a context-free grammar G with $L = L(G)$ such that each word in $L \setminus L'$ is generated by less than k derivation trees.

Obviously, each context-free language L is a complete at least 1-ambiguous subset of L. Let us consider two languages L_1 and L_2 as equivalent if their symmetric difference $((L_1 \setminus L_2) \cup (L_2 \setminus L_1))$ is finite. It can be easily seen that for each context-free language L and each $k \in \mathbb{N}$ the set of complete k-ambiguous subsets of L is either empty or one equivalence class. Using the notion of complete at least k ambiguous subsets one can reveal interesting structures. For instance $\tilde{L}_1 := \{a^n b^n c^n \mid n \in \mathbb{N}\}$ is a complete at least 2-ambiguous subset of $L_1 := \{a^i b^j c^k \mid i = j \text{ or } j = k\}$. As another example we can show that $\tilde{L}_2 := \{a^{2^0} \# a^{2^1} \# \cdots a^{2^n} \mid n \in \mathbb{N}\}$ is a complete at least 2-ambiguous subset of $L_2 := L^* a^* \cup a L^*$, where $L := \{a^n \# a^{2n} \mid n \in \mathbb{N}\}$. The languages L_1 and L_2 both are inherently ambiguous of degree 2. Despite that there is a fundamental structural difference between them: The number of words of length up to n in a complete at least 2-ambiguous subset grows in L_2 by far slower than in L_1. More formally, let $f_1, f_2 : \mathbb{N} \to \mathbb{N}$ be two functions defined by $f_i(n) := |\tilde{L}_i \cap \Sigma^{\leq n}|$ for each $i \in \{1, 2\}$. Then $f_1 \in \Theta(n)$ und $f_2 \in \Theta(\log(n))$. Instead of asking for the maximal number of derivation trees for words of a certain length one can also ask for the number of words requiring an ambiguous generation. The definition above also allows us to combine both points of view. For instance $\tilde{L}_1$ is as well a complete 2- and 3-ambiguous subset of $L_3 := \{a^i b^j c^k \mid i = j \text{ or } j = k \text{ or } k = i\}$, that is L_3 is a 3-ambiguous language, which is generated by a context-free grammar G which generates each word in $\tilde{L}_1$ with 3 derivation trees and the remaining words unambiguously. Hence, it does not generate a single word with ambiguity 2.

It is also interesting to consider the ambiguity of subsets of derivation trees or embedded trees. For instance it has been shown in this thesis that a cycle-free context-free grammar G is exponentially ambiguous if and only if the corresponding set of pumping trees is ambiguous.

It is also possible to consider the combined complexity of a context-free language with respect to different properties. For instance, the language $L := \{a^i b^i a^j b^j \mid i, j \in \mathbb{N}\}$ is as well generated by a linear context-free grammar G_1 and an unambiguous context-free grammar G_2, but among the context-free grammars generating L there is no one which is linear and unambiguous at the same time. Thus, a context-free grammar G generating L either allows a pair of independent internal nodes or G is ambiguous. More generally one can also consider the product of the width of a nonterminal bounded context-free grammar and its degree of ambiguity as a complexity measure. Questions (vi) and (vii) deal with this combined complexity measure.

9.3 Open Problems

Finally, some open problems occurred during the preparation of this thesis are listed. In some cases remarks to their relevance, origin of the problems, or suggestions for their solution are provided. It may be the case that some questions have trivial answers overseen by the author. This is particular likely for questions which occured late in the preparation, when the author had no more time to think them over carefully. This concerns the questions from (vi) on.

(i) Does each context-free language L have a corresponding inherent ambiguity function? The author weakly conjectures that this question should have a positive answer. There are however two scenarios imaginable which would lead to a negative solution:

   (a) There is an infinite sequence of grammars, $G_1, \ldots, G_i, \ldots$ such that $L = L(G_n)$ for each $n \in \mathbb{N}$ and for each $n \in \mathbb{N}$ we have $\hat{d}_{G_{n+1}} \in o^T(\hat{d}_{G_n})$.

   (b) There are two context-free grammars G_1, G_2 such that $L = L(G_1) = L(G_2)$ but their ambiguity functions are incomparable in the sense that $\hat{d}_{G_1} \notin o^T(\hat{d}_{G_2})$ and $\hat{d}_{G_2} \notin o^T(\hat{d}_{G_1})$. An example of a pair of incomparable functions is: $f_e, f_o : \mathbb{N} \to \mathbb{N}$ defined by:

$$f_e(n) := \big(\max\big\{i \in \{2j \mid j \in \mathbb{N}\} \mid i! < n\big\}\big)!$$
$$f_o(n) := \big(\max\big\{i \in \{2j+1 \mid j \in \mathbb{N}\} \mid i! < n\big\}\big)!$$

(ii) Is the set of ambiguity functions of linear context-free grammar a proper subset of the set of ambiguity functions for arbitrary context-free grammars?

(iii) It is possible that the inherent ambiguity function of a linear context-free language L is no ambiguity function of any linear context-free grammar generating L. In such a case the "least" ambiguous context-free grammar generating L is non-linear. It might be more suitable to define a version of inherent ambiguity functions specialised for linear context-free languages. That is we replace each occurence of context-free grammars and languages by linear context-free languages and grammars in the Definitions 2.96 and 2.97, respectively. Let us call the resulting version of inherent ambiguity functions *linearised inherent ambiguity functions*. Now we can reask several questions for linearised inherent ambiguity functions. For instance:

(a) Does each linear context-free language L have a corresponding linearised inherent ambiguity function?

(b) Does the set of linearised inherent ambiguity functions coincide with the set of ambiguity functions of cycle-free linear context-free grammars?

(iv) Is each context-free language with finite ambiguity a finite union of unambiguous context-free languages?

Restricted to so-called bounded languages the answer is positive, i.e., each bounded language with finite degree of ambiguity is a finite union of unambiguous context-free languages [12].

The author only made partial progress on this question: Let G be a context-free grammar and let $P_{<\omega}$ be the set of G's bounded productions. Let us call G a *bottleneck grammar* if $\pi_{P_{<\omega}}(\Delta_G)$ is a singleton, i.e., each derivation tree consists of a fixed sequence of bounded productions with the remaining symbols somehow shuffled inside. A language is a *bottleneck language* if it is generated by some bottleneck grammar. It can be shown that each context-free language can be written as a finite union of bottleneck languages.

(A sketch of the corresponding proof is: Let $G = (N, \Sigma, P, S)$ be a context-free grammar and let $P_{<\omega}$ be the set of G's bounded productions. Obviously, $\pi_{P_{<\omega}}(\Delta_G)$ is finite. For each word $\tau \in \pi_{P_{<\omega}}(\Delta_G)$ we generate a bottleneck grammar G_τ, such that $L(G_\tau)$ generates exactly those words in $L(G)$ which are generated by some derivation tree ρ of G with $\pi_{P_{<\omega}}(\rho) = \tau$. This is done as follows: The nonterminal set of G_τ is the Cartesian product of N and the set of infixes of τ. The infix specifies the sequence of bounded productions occurring in the subtree dominated by the nonterminal. The production set of G_τ contains

copies of each production in $[A, \alpha] \in P \setminus P_{<\omega}$ for each infix γ of τ. The copies are the set of productions of the form $[(A, \gamma), \alpha']$ where α' equals α if we ignore the infixes attached to the nonterminals and the concatenation of the attached infixes is γ. Similarly G_τ contains copies of each production $[A, \alpha] \in P_{<\omega}$ but only for infixes γ of τ which begins with $[A, \alpha]$. Let γ' be the string which remains after cancellation of the first symbol from γ. Now the copies of $[A, \alpha]$ for the infix γ are all the productions of the form $[(A, \gamma), \alpha']$ where α' is α if we ignore the attached infixes and the concatenation of the infixes yield γ'. (Note that G_τ may contain several useless symbols.)

Hence, the question above can be reduced to the question:

Is each bottleneck language with finite ambiguity a finite union of unambiguous context-free languages?

(v) Let Σ be a two letter alphabet. Let $L \subseteq \Sigma^*$ be the set of words with the property that they cannot be transformed into a word of the form w^2 by changing less than two letters. Is L context-free? A more formal specification of L is:

$$\left\{ w \in \Sigma^* \;\middle|\; \begin{array}{c} \exists n \in \mathbb{N} : |w| = 2n \text{ and } \exists i, j \in [1, n] : i \neq j \\ \text{such that } w[i] \neq w[i+n] \text{ and } w[j] \neq w[j+n] \end{array} \right\}$$

Intuitively this question seems to be easier then the famous question whether the set of primitive words Q is context-free. Despite that one faces similar difficulties in attacking this problems. Even if this question has not much to do with this thesis the author likes to raise it.

(vi) Let $n_1, n_2, n \in \mathbb{N}$ such that $n_1 \cdot n_2 \leq n$. Let

$$L_n := \big((a^*ba^*b)^*\{a^iba^i \mid i \in \mathbb{N}\}\big(ba^*ba^*)^*\big) \cap (a^*ba^*)^{2n-1}.$$

It can be shown that L_n is an unambiguous context-free language and a linear context-free language. Moreover, it can be shown that L_n is generated by some nonterminal bounded context-free grammar with a width of n_2 and an ambiguity of n_1 if $n_1 \cdot n_2 \geq n$. Note that this implies that L_n is linear and unambiguous.

Is it possible to generate L_n by any nonterminal bounded context-free grammar with a width of at most n_2 and an ambiguity of at most n_1? (Conjecture: No)

(vii) The language

$$L_* := \big(a^*ba^*b\big)^*\{a^nba^n \mid n > 0\}\big(ba^*ba^*\big)^*$$

is an unambiguous context-free language and a linear context-free language. The linearity is obvious. For the unambiguity note that L_* can be written as:

$$L_* = \{a^nba^m \mid n \neq m\}^*\{a^nba^n \mid n > 0\}\big(a^*ba^*b\big)^*.$$

The languages $\{a^nba^m \mid n \neq m\}$, $\{a^nba^n \mid n > 0\}$, and $\big(a^*ba^*b\big)^*$ are unambiguous. Due to the block structure their concatenation is unambiguous.

Is L_* generated by any nonterminal bounded context-free grammar with a bounded degree of ambiguity. (Conjecture: No)

(viii) Let G be a context-free grammar. A phrase of a word w is an interval $[i, j] \subseteq [1, |w|]$ such that there is a tree $\rho \in \Delta_G$ with $\downarrow(\rho) = w$ and for some $\tau_1, \tau_2 \in T_G^*$ and some $\omega \in \mathit{subtree}(\rho)$ we have $\rho = \tau_1\omega\tau_2$, $|\downarrow(\tau_1)| = i - 1$, and $\downarrow(\tau_1\omega) = j$. A context-free grammar has a unique phrase structure if two phrases for one word are either disjoint or one is a subset of the other. The notion of unique phrase structure has already been introduced by Parikh in [26, Definition 11]. He showed (Theorem 1) that each unambiguous grammar has unique phrase structure. Hence, to prove that a context-free language L is ambiguous it is sufficient to show that each context-free grammar generating it does not have a unique phrase structure. In fact, within all the proofs for inherent ambiguity we examined the least number of trees required to distribute phrases of a word such that each derivation tree has a unique phrase structure. But the converse of Parikh's statement is not true. An example for an ambiguous context-free grammar with unique phrase structure is:

$$S \to AA \mid aa, \quad A \to a.$$

This leads us to the question:

Is there any ambiguous context-free language L such that some context-free grammar G with $L(G) = L$ has a unique phrase structure?

The prove of the ambiguity of such a language L would require different proof techniques then the ones we used in this thesis. The author conjectures that this question has a negative answer. Ambiguities of a context-free grammar G which does not destroy the unique phrase structure does not seem to be fundamental enough to be unavoidable.

Anhang A

Zusammenfassung

A.1 Einleitung

Eine kontextfreie Grammatik G ist eindeutig, wenn jedes Wort höchstens einen Ableitungsbaum hat, andernfalls ist G mehrdeutig. Eine kontextfreie Sprache ist eindeutig, wenn sie von einer eindeutigen kontextfreien Grammatik erzeugt wird. Kontextfreie Grammatiken und Sprachen sind mehrdeutig, wenn sie nicht eindeutig sind. Mehrdeutige kontextfreie Sprachen bezeichnet man auch als inhärent mehrdeutig. Die Existenz mehrdeutiger kontextfreier Sprachen wurde in [26, 27] gezeigt. So gesehen tragen Mehrdeutigkeiten wesentlich zur generativen Mächtigkeit der kontextfreien Grammatiken bei. Mehrdeutigkeit ist für kontextfreie Sprachen und Grammatiken unentscheidbar. (Siehe in [8, 9, 14] oder in den Lehrbüchern [16, Theorem 8.4.5, 8.4.6], [17, Theorem 8.9, 8.16]).

Für eine kontextfreie Grammatik G ist die Mehrdeutigkeit eines Wortes w die Anzahl der Ableitungsbäume mit der w von G generiert wird. Mehrdeutige kontextfreie Sprachen und Grammatiken können nach dem Grad der Mehrdeutigkeit differenziert werden. Dabei heißt eine Grammatik k-deutig, wenn k die kleinste obere Schranke für die Mehrdeutigkeit der erzeugten Wörter ist. Entsprechend heißt eine kontextfreie Sprache k-deutig, wenn sie von einer k-deutigen, aber von keiner $k-1$-deutigen kontextfreien Grammatik erzeugt wird. Für jedes $k \in \mathbb{N}$ gibt es k-deutige kontextfreie Sprachen [23]. Es gibt sogar Sprachen mit einem unendlichen Mehrdeutigkeitsgrad [30]. Ein besonders interessantes Beispiel ist die Crestin Sprache [10]. Dabei handelt es sich um das Quadrat der (eindeutig kontextfreien) Sprache der Palindrome, d. h. um die Menge der Wörter die sich in zwei Palindrome faktorisieren lassen.

Zunächst sieht das nach einer kompletten Antwort auf die Frage nach

möglichen Mehrdeutigkeitsgraden aus. Es stellt sich jedoch in natürlicher Weise die Frage, wie sich, bei unendlichdeutigen kontextfreien Sprachen und Grammatiken, die Mehrdeutigkeit zur Wortlänge verhält. Wahrscheinlich wurde diese Frage erstmals in [16, Section 7.1] angesprochen. Es heißt dort:

„[...] there are inherently ambiguous languages that have an exponential number of derivation trees in the length of the string". [1]

Übersetzt heißt das:

„[...] es gibt inhärent mehrdeutige Sprachen, die eine exponentielle Zahl von Ableitungsbäumen in der Länge der Zeichenketten haben."[2]

Die Mehrdeutigkeitsfunktion $\hat{d}_G$ einer kontextfreien Grammatik ist diejenige monoton nicht fallende Funktion, welche jeder natürlichen Zahl n die kleinste obere Schranke für die Mehrdeutigkeit der Wörter bis zur Länge n zuordnet. Während man für kontextfreie Grammatiken jedem Wort eine Mehrdeutigkeit zuordnen kann, macht eine solche Zuordnung im Fall einer kontextfreien Sprache keinen Sinn, da es für jede kontextfreie Sprache L und jedes Wort w eine kontextfreie Grammatik gibt welche w eindeutig erzeugt. Eine Art, die Mehrdeutigkeit einer kontextfreien Sprache L dennoch nachzuweisen besteht darin, eine unendliche Teilmenge L' anzugeben, so dass jede kontextfreie Grammatik, die L erzeugt, alle bis auf endlich viele Wörter in L' mehrdeutig erzeugt. Dennoch kann manchen kontextfreien Sprachen sinnvoll eine (inhärente) Mehrdeutigkeitsfunktion zugeordnet werden.[3] Eine Funktion f ist eine inhärente Mehrdeutigkeitsfunktion für eine kontextfreie Sprache L, wenn die folgenden beiden Bedingungen erfüllt sind.

(i) Die Sprache L wird von einer f-deutigen kontextfreien Grammatik erzeugt.

(ii) Für jede kontextfreie Grammatik G, die L erzeugt, gibt es eine Konstante $c_G \in \mathbb{N}$, so dass für jedes $n \in \mathbb{N} \setminus 0$ die kleinste obere Schranke für die Mehrdeutigkeit der Wörter bis zur Länge $c_G \cdot n$ mindestens $f(n)$ beträgt.

Grob ausgedrückt bestimmt eine solche Funktion die Länge eines kürzesten Wortes, das eine gegebene Mehrdeutigkeit $d \in \mathbb{N}$ überschreitet, bis auf einen konstanten Faktor in der Wortlänge. Dabei hängt der Faktor nur von der Pumpkonstante der verwendeten kontextfreien Grammatik ab.

[1] Ein korrektes Beispiel für eine solche Sprache wird in [16, Section 7.3] angegeben. Leider weist der zugehörige Beweis eine Lücke auf, die jedoch in [24] geschlossen wurde.

[2] Natürlich ist gemeint, dass alle kontextfreien Grammatiken, welche die Sprache erzeugen, eine exponentielle Anzahl von Ableitungsbäumen in der Wortlänge haben.

[3] Es ist tatsächlich nicht klar, ob jede kontextfreie Sprache eine Mehrdeutigkeitsfunktion gemäß der folgenden Definition hat.

Diese Dissertation untersucht die Mehrdeutigkeit kontextfreier Sprachen und Grammatiken. Ein Ziel ist dabei, eine möglichst genaue Charakterisierung der Menge der Mehrdeutigkeitsfunktionen für kontextfreie Sprachen und Grammatiken zu finden. Desweiteren werden Trennungen verschiedener Mehrdeutigkeitsklassen sowie Beziehungen zwischen denselben untersucht. Schließlich werden Zusammenhänge zwischen Mehrdeutigkeit und dem Aufwand zur Worterkennung herausgearbeitet.

A.2 Grundbegriffe

Sei G eine kontextfreie Grammatik. Ein Symbol (Terminal oder Nichtterminal) von G heißt *nutzlos*, wenn es in keinem Ableitungsbaum auftauchen kann. Dabei verlangen wir von Ableitungsbäumen, dass sie vollständig sind, d. h., die Wurzel ist mit dem Startsymbol und die Blätter sind ausschließlich mit Terminalen beschriftet. Eine kontextfreie Grammatik heißt *reduziert*, wenn sie keine nutzlosen Symbole enthält. Offenbar haben nutzlose Symbole keinen Einfluss auf die Mehrdeutigkeit einer kontextfreien Grammatik. Es genügt daher, reduzierte kontextfreie Grammatiken zu betrachten.

Die Grammatik G ist *zyklisch*, wenn sie ein Nichtterminal hat, welches sich selbst in einer nicht leeren Folge von Ableitungsschritten generiert, d. h. $A \overset{+}{\Rightarrow}_G A$. Die Mehrdeutigkeitsfunktion f einer reduzierten kontextfreien Grammatik G hat für fast alle (alle bis auf endlich viele) Argumente den Wert ∞ genau dann, wenn G zyklisch ist. Da jede kontextfreie Grammatik in eine äquivalente zykelfreie kontextfreie Grammatik transformiert werden kann, sind Mehrdeutigkeitsfunktionen, die den Wert ∞ erreichen, für keine kontextfreie Sprache inhärent. Deshalb betrachten wir im Folgenden nur reduzierte zykelfreie kontextfreie Grammatiken.

A.3 Ergebnisse

Offenbar gibt es mehrdeutige kontextfreie Grammatiken, die eindeutig kontextfreie Sprachen erzeugen. In diesem Sinne kann die Mehrdeutigkeit einer Grammatik überflüssig sein. Nun stellt sich aber die Frage, ob es Mehrdeutigkeitsfunktionen kontextfreier Grammatiken gibt, die für keine kontextfreie Sprache inhärent sind. Dies ist aber nicht der Fall:

Satz A.1 *Die Menge der Mehrdeutigkeitsfunktionen kontextfreier Sprachen und zykelfreier kontextfreier Grammatiken sind identisch.*

Das heißt, gleichgültig wie künstlich die Mehrdeutigkeitsfunktion $\hat{d}_G$ einer kontextfreien Grammatik G anmuten mag, es gibt stets eine kontextfreie Sprache L_G, so dass L_G inhärent $\hat{d}_G$-deutig ist. Im Beweis wird aus G auf Basis der Ableitungsbaummenge Δ_G eine kontextfreie Sprache L effektiv konstruiert, die nicht generiert werden kann, ohne die Struktur von Δ_G nachzubilden.

Satz A.1 reduziert die Frage, ob eine gegebene Funktion f eine inhärente Mehrdeutigkeitsfunktion einer kontextfreien Sprache ist, auf die entsprechende Frage für kontextfreie Grammatiken. Um zu zeigen, dass eine bestimmte Mehrdeutigkeitsfunktion $f : \mathbb{N} \to \mathbb{N}$ für eine kontextfreie Sprache inhärent ist, genügt es also, eine f-deutige kontextfreie Grammatik G anzugeben. Dabei ist die Mehrdeutigkeit der von L erzeugten Sprache unerheblich.

Nun wenden wir uns der Frage zu, welche Mehrdeutigkeitsfunktionen es für zykelfreie kontextfreie Grammatiken tatsächlich gibt. Es ist leicht, für jedes $k \in \mathbb{N}$ rechtslineare Grammatiken über einem Alphabet mit nur einem Element anzugeben, die k-deutig, $\Theta(n^k)$-deutig bzw. $2^{\Theta(n)}$-deutig sind. Gemäß Satz A.1 reichen diese einfachen Beispiele trotz ihrer Regularität bereits hin, um die Existenz der entsprechenden inhärenten Mehrdeutigkeiten zu zeigen. Dies vereinheitlicht die Beweisführung für die Existenz der in [23, 30, 24] bereits gezeigten inhärenten Mehrdeutigkeiten. Mithilfe von [16, Theorem 12.2.1] sieht man leicht, dass die Mehrdeutigkeit zykelfreier kontextfreier Grammatiken durch $2^{\mathcal{O}(n)}$ beschränkt ist.

Sei G eine kontextfreie Grammatik. Ein Teilbaum eines Ableitungsbaums von dem ein Teilbaum abgeschnitten wurde heißt Pumpbaum von G, falls er ein Blatt hat, dass mit dem gleichen Nichtterminal wie die Wurzel beschriftet ist. Der Begriff des Pumpbaums ist zentral für den Beweis der meisten Pumpinglemmata. Es wird gezeigt, dass:

Lemma A.2 *Eine zykelfreie kontextfreie Grammatik ist exponentiell mehrdeutig, wenn die Menge ihrer Pumpbäume mehrdeutig ist, d. h., wenn es zwei Pumpbäume mit gleicher Front gibt.*

Eine Produktion p einer kontextfreien Grammatik G heißt beschränkt, wenn es eine Konstante $k \in \mathbb{N}$ gibt, so dass keine Ableitung aus dem Startsymbol mehr als k Anwendungen von p enthält.

Lemma A.3 *Eine zykelfreie kontextfreie Grammatik G heißt polynomiell mehrdeutig, wenn man durch Einfügen von Markierungen in die rechten Regelseiten beschränkter Produktionen eine eindeutige kontextfreie Grammatik erhalten kann.*

Die Kriterien in den beiden vorangegangenen Lemmata sind jeweils hinreichend für die zugehörige Mehrdeutigkeitsklasse und schließen sich gegenseitig aus, d. h., jede kontextfreie Grammatik kann nur eines der beiden Kriterien erfüllen. Tatsächlich erfüllt jede zykelfreie kontextfreie Grammatik eines der beiden Kriterien, d. h.:

Lemma A.4 *In einer zykelfreien kontextfreien Grammatik, deren Pumpbaummenge eindeutig ist, lassen sich die beschränkten Produktionen so markieren, dass die resultierende Grammatik eindeutig ist.*

Aus den vorangegangenen drei Lemmata ergibt sich unmittelbar:

Satz A.5 *Für jede kontextfreie Grammatik G kann effizient eine Konstante $k \in \mathbb{N}$ mit der Zusicherung errechnet werden, dass die Mehrdeutigkeitsfunktion von G entweder in $2^{\Theta(n)}$ oder in $\mathcal{O}(n^k)$ liegt. Eine (zykelfreie) kontextfreie Grammatik ist stets entweder exponentiell (d. h. in $2^{\Theta(n)}$) oder polynomiell beschränkt (d. h. in n^k).*

Satz A.5 wurde bereits in der Diplomarbeit des Autors bewiesen, allerdings durch einen technischen Beweis, der zeigt, dass das Kriterium aus A.2 notwendig für exponentielle Mehrdeutigkeit ist. Lemma A.3 und Lemma A.4, die neu sind, zerlegen diese Beweisführung in zwei leicht verständliche Teilziele. Ein wesentlicher Erkenntnisfortschritt liegt hier in der Herausarbeitung der besonderen Rolle der beschränkten Produktionen, die im alten Beweis gar nicht vorkamen. Die Erkenntnis, dass bestimmte Produktionen eine für polynomielle Mehrdeutigkeit herausragende Rolle spielen, vertieft und erleichtert das Verständnis des Sachverhalts erheblich. Darüber hinaus kommt man auf Grundlage der neuen Herangehensweise meist zu einer wesentlichen Verbesserung der in Satz A.5 zugesicherten Konstante k. (Von j^n, wobei j die maximale Anzahl der Nichtterminale auf der rechten Seite einer Produktion und n die Anzahl der Nichtterminale ist, auf die maximale Anzahl von Markierungen m in den Wörtern der markierten Grammatik. Dabei ist m effizient berechenbar und es gilt $m \leq j^n$.)

Darüber hinaus ergibt sich aus Lemma A.3 unmittelbar eine wichtige Beziehung zwischen der Klasse der eindeutigen kontextfreien Grammatiken *UCFL* und der Klasse der kontextfreien Grammatiken mit polynomiell beschränkter Mehrdeutigkeit *PCFL*:

Satz A.6 *Die Klasse der kontextfreien Sprachen mit polynomiell beschränkter Mehrdeutigkeit $PCFL$ fällt mit dem Abschluss der eindeutigen kontextfreien Sprachen $UCFL$ unter beschränkten Kontraktionen zusammen.*

Bei beschränkten Kontraktionen handelt es sich um Projektionen, die nur Symbole löschen dürfen, deren Anzahl von Vorkommen in beliebigen Wörtern der Sprache durch eine globale Konstante beschränkt ist. Dadurch lässt sich ein Resultat von Rossmanith und Rytter von *UCFL* auf *PCFL* generalisieren.

Korollar A.7 *Jede kontextfreie Sprache mit polynomiell beschränkter Mehrdeutigkeit lässt sich auf einer CREW-PRAM*[4] *in logarithmischer Zeit erkennen.*

Ein weiteres interessantes Resultat ist die Existenz von sublinearer Mehrdeutigkeit. Für logarithmische und quadratwurzelförmige Mehrdeutigkeit wurden in [33] linear kontextfreie Sprachen explizit angegeben. Tatsächlich können wir zeigen:

Lemma A.8 *Für jede berechenbare totale monoton nicht fallende divergente Funktion f gibt es eine kontextfreie Grammatik G mit unendlicher Mehrdeutigkeit, deren Mehrdeutigkeitsfunktion langsamer wächst als f, d. h. $\forall n \in \mathbb{N} : \hat{d}_G(n) \leq f(n)$.*

Unter Anwendung von Satz A.1 ergeben sich die folgenden Resultate:

Theorem A.9 *Für jede berechenbare totale monoton nicht fallende divergente Funktion f gibt es eine kontextfreie Sprache L mit unendlicher Mehrdeutigkeit, mit einer Mehrdeutigkeitsfunktion $\hat{d}_L$, die langsamer wächst als f, d. h. $\forall n \in \mathbb{N} : \hat{d}_G(n) \leq f(n)$.*

Theorem A.10 *Seien $f, g : \mathbb{N} \to \mathbb{N}$ zwei inhärente Mehrdeutigkeitsfunktionen. Dann gibt es eine inhärente Mehrdeutigkeitsfunktion in $\Theta^T((f \cdot g)(n))$.*

Tatsächlich lässt sich dies sogar mit speziellen linearen Grammatiken erzielen. Da jede Mehrdeutigkeit - wie oben erwähnt - inhärent für eine kontextfreie Sprache ist, gibt es für jede solche Funktion f eine kontextfreie Sprache mit unbeschränktem Mehrdeutigkeitsgrad, deren inhärente Mehrdeutigkeitsfunktion langsamer als f wächst. Wir können also zum Beispiel eine unbeschränkt mehrdeutige kontextfreie Sprache finden, deren inhärente Mehrdeutigkeitsfunktion unterhalb von $\log^*$ angesiedelt ist. Kontextfreie Grammatiken mit sublinearer Mehrdeutigkeit erweisen sich im Zusammenhang mit dem Parsing Algorithmus von Earley [11] als interessant. Es gilt:

[4]Eine CREW-PRAM ist eine Random-Access-Maschine mit einer polynomiellen Anzahl von Prozessoren (PRAM). Der gemeinsame Speicher erlaubt es mehreren Prozessoren im gleichen Zeitschritt auf eine Speicherstelle lesend zuzugreifen (concurrent read (CR)), aber es darf nur ein Prozessor zu einer bestimmten Zeit eine Speicherzelle überschreiben (exclusive write (EW)).

Satz A.11 *Betrachtet man die Größe der untersuchten Grammatik als eine Konstante und bestimmt die Laufzeit nur in Abhängigkeit von der Wortlänge, so liegt die Zeitkomplexität des Wortproblems mit Earley's Algorithmus (sequentiell) innerhalb von $\mathcal{O}(n^2 \cdot \hat{d}_G(n + k_G))$, wobei $\hat{d}_G$ die Mehrdeutigkeitsfunktion von G und $k_G \in \mathbb{N}$ eine nur von G abhängige Konstante ist.*

Die Konstante k_G im vorangegangenen Satz kann weggelassen werden, wenn die Mehrdeutigkeitsfunktion von G eine sehr schwache Homogenitätsbedingung erfüllt. (Dem Autor ist keine Mehrdeutigkeitsfunktion bekannt, welche diese Homogenitätsbedingung nicht erfüllt.)

A.4 Ausblick

Bisher gibt es keine vollständige nichttriviale Charakterisierung der Menge der Mehrdeutigkeitsfunktionen. Aus Theorem 8.1 ergibt sich, dass es nicht notwendig ist, diese Frage getrennt für zykelfreie kontextfreie Grammatiken und kontextfreie Sprachen zu behandeln. Ferner kann man den Bereich, in dem Mehrdeutigkeitsfunktionen liegen können, erheblich einschränken:

(i) Es gibt keine Mehrdeutigkeitsfunktion in $\omega(2^{\mathcal{O}(n)})$.

(ii) Es gibt keine Mehrdeutigkeitsfunktion in $o(2^{\Omega(n)}) \cap \omega(n^{\mathcal{O}(1)})$.

(iii) Gemäß Satz A.9 gibt es Mehrdeutigkeitsfunktionen, die so langsam wie eine beliebige berechenbare Funktion wachsen.

(iv) Seien f und $g : \mathbb{N} \to \mathbb{N}$ zwei inhärente Mehrdeutigkeitsfunktionen. Dann gibt es eine inhärente Mehrdeutigkeitsfunktion in $\Theta^T((f \cdot g)(n))$.

Dennoch klärt dies nicht, welche Mehrdeutigkeitsfunktionen es genau gibt und in welcher Dichte sie auftreten. Satz A.9 besagt lediglich, dass substantiell unterhalb einer divergenten Mehrdeutigkeitsfunktion es stets weitere divergente Mehrdeutigkeitsfunktionen gibt. Er sagt hingegen nichts darüber aus, ob es weitere Lücken wie die zwischen polynomieller und exponentieller Mehrdeutigkeit im sublinearen Bereich gibt.

Im Beweis von Satz A.8 wurde zu einer beliebigen Turing Maschine M eine kontextfreie Grammatik G konstruiert, welche im Wesentlichen (jede zweite Konfiguration ist gespiegelt) eine Obermenge der Menge der Berechnungsfolgen von M generiert. Die Mehrdeutigkeitsfunktion wird dabei bereits von den gültigen Berechnungsfolgen eindeutig festgelegt, da nur diese kürzeste Wörter zu einer gegebenen Mehrdeutigkeit sein können. Die Mehrdeutigkeit einer Berechnungsfolge war dabei einfach die Häufigkeit, mit der ein bestimmter Zustand erreicht wird. Diese Konstruktion legt den Verdacht nahe, dass

man die Mehrdeutigkeit im sublogarithmischen Bereich sehr fein justieren und damit gute Dichteresultate erzielen kann. Jedoch ist die Abschätzung der Länge von Berechnungsfolgen technisch schwierig. Hier fließt sowohl die Zeitkomplexität über die Anzahl der Konfigurationsübergänge als auch die Platzkomplexität über die Länge der einzelnen Konfigurationen ein. Gewissermaßen handelt es sich um den, über die Zeit aufsummierten, Platzverbrauch. Andererseits ist man nicht notwendig an Turing Maschinen als Berechnungsmodell gebunden. Statt der Übergangsrelation einer Turingmaschine, kann man wesentlich allgemeinere eindeutig kontextfreie Relationen zulassen. Dabei heißt eine Relation $R \subseteq \Sigma \times \Sigma$ für ein Alphabet Σ eindeutig kontextfrei, wenn $L := \{u\#v^R \mid (u,v) \in R\}$ eindeutig kontextfrei ist, wobei $\# \notin \Sigma$ und v^R die Spiegelung von v ist.

Ein weiteres Forschungsfeld ist die Untersuchung der für die Mehrdeutigkeit verantwortlichen Teilmengen. Obwohl es kein Wort in einer kontextfreien Sprache L gibt, das von allen kontextfreien Grammatiken, die L erzeugen, mehrdeutig generiert wird, lassen sich unendliche Teilmengen der erzeugten Sprache für die Mehrdeutigkeit von L gemäß der folgenden Definition verantwortlich machen.

Definition A.12 *Sei L eine kontextfreie Sprache, und $k \in \mathbb{N}$. Eine Sprache $L' \subseteq L$ ist eine* vollständige mindestens k-deutige Teilmenge *von L, falls folgende Bedingungen erfüllt sind:*

(i) Jede kontextfreie Grammatik G mit $L = L(G)$ erzeugt alle bis auf endlich viele Wörter in L' mit mindestens k Ableitungsbäumen.

(ii) Es gibt eine kontextfreie Grammatik G mit $L = L(G)$, für die jedes Wort aus $L \setminus L'$ weniger als k Ableitungsbäume hat.

Offenbar ist jede kontextfreie Sprache L mindestens 1-deutig in L. Mit diesem Begriff lassen sich interessante Strukturen entdecken: Zum Beispiel kann man zeigen, dass die Sprache $\tilde{L}_1 := \{a^n b^n c^n \mid n \in \mathbb{N}\}$ eine vollständige mindestens 2-deutige Sprache in $L_1 := \{a^i b^j c^k \mid i = j \text{ oder } j = k\}$ ist. Man kann auch zeigen, dass $\tilde{L}_2 := \{a^{2^0}\#a^{2^1}\#\cdots a^{2^n} \mid n \in \mathbb{N}\}$ eine vollständige mindestens 2-deutige Sprache in $L_2 := L^*a^* \cup aL^*$ ist, wobei $L := \{a^n\#a^{2n} \mid n \in \mathbb{N}\}$ ist. Sowohl L_1 als auch L_2 sind 2-deutig. Dennoch gibt es einen interessanten Unterschied zwischen L_1 und L_2, nämlich den, dass L_2 wesentlich weniger mehrdeutige Wörter zu einer gegebenen Wortlänge besitzt als L_1. Man betrachte dazu die Funktionen $f_1, f_2 : \mathbb{N} \to \mathbb{N}$ definiert durch $f_i(n) := |\tilde{L}_i \cap \Sigma^{\leq n}|$ für $i \in \{1,2\}$. Es gilt $f_1 \in \Theta(n)$ und $f_2 \in \Theta(\log(n))$. Statt zu fragen *wieviele Ableitungsbäume* es für Wörter einer gewissen Länge gibt, kann man auch fragen, *wieviele Wörter* bis zu einer gewissen Länge mehrdeutig erzeugt werden. Man kann natürlich auch beide Ansätze kombinieren. Zum Beispiel ist $\tilde{L}_1$

sowohl eine vollständige mindestens 2-deutige, als auch eine vollständige mindestens 3-deutige Teilmenge in $L_3 := \{a^i b^j c^k \mid i = j \text{ oder } j = k \text{ oder } k = i\}$, d. h., dass L_3 eine 3-deutige Sprache ist, die von einer kontextfreien Grammatik erzeugt wird, welche Wörter entweder eindeutig oder 3-deutig, aber nicht 2-deutig generiert.

Es kann auch erhellend sein, Teilmengen der Menge der Ableitungsbäume oder Teilstrukturen derselben zu untersuchen. Beispielsweise wird in dieser Arbeit gezeigt, dass eine zykelfreie kontextfreie Grammatik G genau dann exponentiell mehrdeutig ist, wenn die Pumpbaummenge von G mehrdeutig ist. Aber es lassen sich auch interessante Bezüge zu linearen Grammatiken herstellen. So lässt sich zeigen, dass die Sprache $L := \{a^i b^i a^j b^j \mid i, j \in \mathbb{N}\}$ sowohl von einer linear kontextfreien Grammatik G_1 als auch von einer eindeutig kontextfreien Grammatik G_2 erzeugt werden kann, es aber keine sowohl lineare als auch eindeutig kontextfreie Grammatik gibt, die L erzeugt. Eine kontextfreie Grammatik muss also entweder unabhängige innere Knotenpaare (also Paare, bei denen keine Komponente Vorgänger der anderen ist) oder Mehrdeutigkeit zulassen, um L zu erzeugen.

A.5 Offene Fragen

Wir haben gesehen, dass es zu jeder berechenbaren totalen monoton nicht fallenden divergenten Funktion f eine kontextfreie Grammatik G mit unendlicher Mehrdeutigkeit gibt, deren Mehrdeutigkeitsfunktion langsamer wächst als f. Somit lassen sich unendliche Folgen kontextfreier Grammatiken mit echt absteigender Mehrdeutigkeit definieren. Dies wirft die Frage auf, ob es zu jeder kontextfreien Sprache eine inhärente Mehrdeutigkeitsfunktion gibt. Wenn nein, dann würde das heißen, dass es eine kontextfreie Sprache L mit unendlicher Mehrdeutigkeit gibt, für die es zu jeder kontextfreien Grammatik G_1 mit $L = L(G_1)$ eine substantiell weniger mehrdeutige kontextfreie Grammatik G_2 mit $L = L(G_2)$ gibt, wobei „substantiell“bedeutet, dass $\hat{d}_{G_2}(n) \leq \hat{d}_{G_1}(n)$ für alle $n \in \mathbb{N}$ gilt und diese Funktionen sich an mehr als endlich vielen Stellen unterscheiden.

Bisher ist keine Mehrdeutigkeitsfunktion bekannt, die nicht auch für eine linear kontextfreie Sprache inhärent ist. Es stellt sich daher die Frage, ob die Menge der inhärenten Mehrdeutigkeitsfunktionen linear kontextfreier Sprachen mit der Menge der inhärenten Mehrdeutigkeitsfunktionen kontextfreier Sprachen zusammenfällt.

In diesem Zusammenhang stellt sich auch die Frage, ob es zu jeder linear kontextfreien Grammatik G eine inhärent $\hat{d}_G$-deutige lineare kontextfreie Sprache gibt. Das geht nicht aus Satz A.1 hervor, da die zugehörige Kon-

struktion nicht linearitätserhaltend ist. Man beachte hierbei, dass es eine linear kontextfreie Sprache L mit einer inhärenten Mehrdeutigkeitsfunktion f geben kann, so dass jede linear kontextfreie Grammatik, die L erzeugt, mehr als f-deutig ist. Dies resultiert aus der Definition der inhärenten Mehrdeutigkeitsfunktion, welche nur die Existenz einer f-deutigen kontextfreien Grammatik, nicht jedoch die einer linearen kontextfreien Grammatik fordert. Für die Untersuchung linear kontextfreier Sprachen erscheint es daher sinnvoller, einen modifizierten Inhärenzbegriff zu benutzen, der analog zu dem in dieser Dissertation benutzten ist, in dem aber konsequent kontextfrei durch linear kontextfrei ersetzt wird. Für diesen Begriff stellen sich die vorangegangenen Fragen neu.

Bibliography

[1] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*, volume I. Prentice-Hall, Englewood Cliffs, N.J., 1972.

[2] Jean-Michel Autebert, Jean Berstel, and Luc Boasson. Context-free languages and pushdown automata. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 111–174. Springer, Berlin-Heidelberg-New York, 1997.

[3] Brenda S. Baker and Ronald V. Book. Reversal-bounded multipushdown machines. *Journal of Computer and System Sciences*, 8:315–332, 1974.

[4] Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Z. Phonetik. Sprachwiss. Kommunikationsforschung*, 14:143–172, 1959.

[5] Alberto Bertoni, Christian Choffrut, Massimiliano Goldwurm, and Violetta Lonati. On the number of occurrences of a symbol in words of regular languages. *Theoretical Computer Science*, 302:431–456, 2002.

[6] Alberto Bertoni, Massimiliano Goldwurm, G. Mauri, and N. Sabadini. Counting techniques for inclusion, equivalence and membership problems. In Volker Diekert and G. Rozenberg, editors, *The Book of Traces*. World Scientific, Singapore, New Jersey, London, Hong Kong, 1995.

[7] Ronald V. Book and Friedrich Otto. *String-rewriting system*. Texts and monographs in computer science. Springer, New York [u.a.], 1993.

[8] D. Cantor. On the ambiguity problem of backus systems. *Journal of the Association for Computing Machinery*, 9:477–479, 1962.

[9] Noam Chomsky and Marcel Paul Schützenberger. The algebraic theory of context-free languages. In *Comp. Prog. and Formal Systems*, pages 118–161, Amsterdam, 1963. North-Holland.

[10] J.P. Crestin. Un langage non ambigu dont le carré est d'ambiguité non bornée. In M. Nivat, editor, *Automata, Languages and Programming*, pages 377–390. Amsterdam, North-Holland, 1973.

[11] Jay Clark Earley. *An efficient context-free parsing algorithm.* PhD thesis, Carnegie-Mellon University, 1968.

[12] Seymour Ginsburg. The *mathematical theory of context free languages.* McGraw-Hill, New York [u.a.], 1966.

[13] Seymour Ginsburg, Sheila A. Greibach, and Michael A. Harrison. One-way stack automata. *Journal of the Association for Computing Machinery*, 14:389–418, 1967.

[14] Seymour Ginsburg and Joseph Ullian. Ambiguity in context free languages. *Journal of the Association for Computing Machinery*, 13(1):62–89, 1966.

[15] Seymour Ginsburg and Joseph Ullian. Preservation of unambiguity and inherent ambiguity in context-free languages. *Journal of the Association for Computing Machinery*, 13(3):364–368, 1966.

[16] Michael A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, 1978.

[17] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[18] Roberto Incitti. The growth function of context-free languages. *Theoretical Computer Science*, 255(1-2):601–605, 2001.

[19] T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report Scientific Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Mass., 1965.

[20] Rainer Kemp. Mehrdeutigkeit kontextfreier Sprachen. In *Lecture Notes in Computer Science*, volume 14 of *Lecture Notes in Computer Science*, pages 534–546, Berlin-Heidelberg-New York, 1974. Springer.

[21] Yuji Kobayashi. Personal communication, 1999.

[22] Werner Kuich. Semirings and formal power series. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 609–677. Springer, Berlin-Heidelberg-New York, 1997.

[23] H. Maurer. The existence of context-free languages which are inherently ambiguous of any degree. Research series, Department of Mathematics, University of Calgary, 1968.

[24] M Naji. Grad der Mehrdeutigkeit kontextfreier Grammatiken und Sprachen, 1998. Diplomarbeit, FB Informatik, Johann–Wolfgang–Goethe–Universität.

[25] William Ogden. A helpful result for proving inherent ambiguity. *Mathematical Systems Theory*, 2(3):191–194, 1968.

[26] R. J. Parikh. Language–generating devices. In *Quarterly Progress Report*, volume 60, pages 199–212. Research Laboratory of Electronics, M.I.T, 1961.

[27] R. J. Parikh. On context-free languages. *Journal of the Association for Computing Machinery*, 13:570–581, 1966.

[28] Peter Rossmanith and Wojciech Rytter. Observations on $\log(n)$ time parallel recognition of unambiguous cfl's. *Information Processing Letters*, 44:267–272, 1992.

[29] Arto Salomaa and Matti Soittola. *Automata theoretic aspects of formal power series*. Springer, 1978.

[30] Eliahu Shamir. Some inherently ambiguous context-free languages. *Information and Control*, 18:355–363, 1971.

[31] Klaus Wich. Kriterien für die Mehrdeutigkeit kontextfreier Grammatiken, 1997. Diplomarbeit, FB Informatik, Johann–Wolfgang–Goethe–Universität.

[32] Klaus Wich. Exponential ambiguity of context-free grammars. In G. Rozenberg and W. Thomas, editors, *Proceedings of the DLT, 1999*, pages 125–138. World Scientific, Singapore, 2000.

[33] Klaus Wich. Sublinear ambiguity. In Mogens Nielsen and Branislav Rovan, editors, *Proceedings of the MFCS 2000*, number 1893 in Lecture Notes in Computer Science, pages 690–698, Berlin-Heidelberg-New York, 2000. Springer.

[34] Klaus Wich. Characterization of context-free languages with polynomially bounded ambiguity. In Jiří Sgall, Aleš Pultr, and Petr Kolman,

editors, *Proceedings of the MFCS 2001, Mariánské Lázně, (Czech Republic), 2001*, number 2136 in Lecture Notes in Computer Science, pages 703–714, Berlin-Heidelberg-New York, 2001. Springer.

[35] Klaus Wich. Universal inherence of cycle-free context-free ambiguity functions. In Peter Widmayer et al., editors, *Proceedings of the ICALP 2002*, number 2380 in Lecture Notes in Computer Science, pages 669–680, Berlin-Heidelberg-New York, 2002. Springer.

[36] D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10:2:189–208, 1967.

Appendix B

Glossar

The glossar lists symbolic notations. Some notations are overloaded, i.e., their meaning depends on the type of subscripts or superscripts. Therefore, in the table of objects we define some objects of different type which are needed to specify the objects in the table of symbolic notations uniquely. Unfortunately, the definition of an object x in the table of Objects may contain objects y_t typed by an object t. In such a case the definition of t occurs earlier than x in the table of objects. Hence, the lookup procedure has no cycles.

B.1 Table of Objects

object	page	type
R, R_1, R_2		relations
M	18	monoid
i, j, n		$i, j, n \in \mathbb{N}$
Σ, Γ		finite alphabets
w		$w \in \Sigma^*$
S		set
G	42	context-free grammar
P	42	production set of G
N	42	nonterminal set of G
Σ	42	terminal set of G
A, B	42	elements of N
X, Y	42	elements of $(N \cup \Sigma)^*$
Z	42	element of $T_G = (N \cup \Sigma \cup P)^*$
L, L_1, L_2	19	formal languages
p	42	production of G

object	page	type
τ		$\tau \in \Gamma^*$
L_Δ	41	tree language
ρ		$\rho \in \mathit{embedded}(\Delta_G)$
$\mathcal{X}$		set of multisets
a		$a \in \Sigma$
f		function $\mathbb{N} \to \mathbb{R}_+$
$g : \mathbb{N} \to \mathbb{N}$		divergent non-decreasing function
ϑ_1, ϑ_2	131	$\vartheta_1, \vartheta_2 \in \Lambda_G^A$

B.2 Table of Symbolic Notations

object	page	no	explanation
$\mathbb{R}$	15	Sec 2.1.1	real numbers
$\mathbb{R}_+$	15	Sec 2.1.1	set of positive real numbers $\mathbb{R}_+ := \{x \in \mathbb{R} \mid x > 0\}$
$\mathbb{N}$	15	Sec 2.1.1	non-negative integers
$\dot\cup$	15	Sec 2.1.1	disjoint union
sR_1tR_2u	16	Sec 2.1.3	$(s,t) \in R_1$ and $(t,u) \in R_2$
$R_1 \circ R_2$	16	Sec 2.1.3	$R_2 \circ R_1 := \{(s,u) \mid \exists t : sR_1tR_2u\}$
R^n	16	Sec 2.1.3	R^0 identity, $R^n := R^{n-1} \circ R$
R^+	16	Sec 2.1.3	$\bigcup_{i=1}^{\infty} R^i$
R^*	16	Sec 2.1.3	$R^+ \cup R^0$
M^+	18	Sec 2.1.5	semigroup generated by M
M^*	18	Sec 2.1.5	monoid generated by M
R^{-1}	16	Sec 2.1.3	$\{(y,x) \mid (x,y) \in R\}$ (inverse relation)
ε	18	Sec 2.1.5	empty word
$[i,j]$	19	Sec 2.1.6	$\{i, \ldots, j\}$ (intervall)
$w[i]$	19	Sec 2.1.6	i-th symbol of w
$\lvert w\rvert$	19	Sec 2.1.6	length of w
$w[i,j]$	19	Sec 2.1.6	$w[i]w[i+1]\cdots w[j]$ (infix)
$\vec{w}$	21	Def 2.4	Parikh vector
2^S	19	Sec 2.1.6	$\{s \mid s \subseteq S\}$ (power set)
$X \vdash Y$	50	Def 2.65	$\exists \rho \in \mathit{embedded}(\Delta_G) : \uparrow(\rho) = X$ and $Y \in \vec{\rho}$
$X \vdash_1 Y$	51	Sec 2.3.8	$\exists p \in P : \ell(p) = X$ and $Y \in r(p)$.
$X \equiv Y$	53	Def 2.70	$X \vdash Y$ and $Y \vdash X$
∇_X	50	Def 2.65	$\{Y \in N \cup \Sigma \mid Y \vdash X\}$
$\mathit{dist}(Y,X)$	55	Def 2.74	Max number of equiv. classes on a path from Y to X in the dependency graph.
$L_1 \cdot L_2$	19	Sec 2.1.6	$\{uv \mid u \in L_1 \ \wedge \ v \in L_2\}$ (concatenation)

object	page	no	explanation
L_1L_2	19	Sec 2.1.6	$L_1 \cdot L_2$ (concatenation)
L^n	19	Sec 2.1.6	$L^0 := \{\varepsilon\}$, $L^n := L \cdot L^{n-1}$
$\Sigma^{\leq n}$	19	Sec 2.1.6	$\cup_{i=0}^{n}\Sigma^i$
$\Sigma^{<n}$	19	Sec 2.1.6	$\cup_{i=0}^{n-1}\Sigma^i$
$\Sigma^{\geq n}$	19	Sec 2.1.6	$\Sigma^* \setminus \Sigma^{<n}$
$L^{\times n}$	19	Sec 2.1.6	$\{(w_1,\ldots,w_n) \mid w_1,\ldots,w_n \in L\}$ (Cartesian product)
$(a_1 \cdots a_n)^R$	19	Sec 2.1.6	$a_n \cdots a_1$ (reversal)
L^R	19	Sec 2.1.6	$\{w^R \mid w \in L\}$ (reversal)
$\pi_\Sigma(w)$	20	Sec 2.1.6	projection
$[a/L]$	20	Def 2.1	substitution
$\sigma(L)$	20	Sec 2.1.6	substitution
P_Γ	26	Def 2.6	$\Gamma \times \Gamma^*$ (internal symbols)
T_Γ	26	Def 2.6	$\Gamma \cup P_\Gamma$ (tree alphabet)
$\rightarrow_\Gamma$	26	Def 2.7	tree expansion
$\leftarrow_\Gamma$	27	Def 2.9	$(\rightarrow_\Gamma)^{-1}$ (tree reduction)
$\leftrightarrow_\Gamma$	29	Def 2.15	$\rightarrow_\Gamma \cup \leftarrow_\Gamma$ (tree transformation)
$\ell(Z)$	30	Def 2.17	left-hand side
$r(Z)$	30	Def 2.17	right-hand side
T_G	42	Def 2.49	$N \cup \Sigma \cup P$ (tree alphabet)
Δ_Γ^τ	27	Def 2.8	forest derived by τ
Δ_Γ	30	Def 2.18	$\cup_{A\in\Gamma}\Delta_\Gamma^A$ (trees)
Δ_G	42	Def 2.49	set of derivation trees
Δ_L	65	Def 2.98	$\{\Delta_G \mid G$ context-free $L = L(G)\}$
$\mathit{subtree}(L_\Delta)$	41	Def 2.47	subtree
$\mathit{cut}(L_\Delta)$	41	Def 2.47	cut of trees
$\mathit{embedded}(L_\Delta)$	41	Def 2.47	embedded trees
$\tilde{\Delta}_G$	136	Def 7.18	trees in $\mathit{cut}(\Delta_G)$ without void symbols
Λ_G	42	Def 2.49	pumping trees
Λ_G^A	131	Def 7.1	A-pumping trees
$\vartheta_1 \odot \vartheta_2$	131	Def 7.2	pumping tree concatenation
$\uparrow(\rho)$	29	Def 2.14	root
$\downarrow(\rho)$	39	Def 2.34	frontier
$\updownarrow(\rho)$	60	Def 2.85	$[\uparrow(\rho),\downarrow(\rho)]$ (interface)
$\mathit{parse}(\rho)$	157	Sec 8.1	$\pi_{N\cup P}(\rho)$
$w(\tau)$	38	Def 2.31	weight of a tree string
w_X	55	Def 2.75	X-weight
$L(G)$	42	Def 2.49	$\{\downarrow(\rho) \mid \rho \in \Delta_G\}$ (generated language)

object	page	no	explanation
$\mathcal{S}_G$	42	Def 2.49	$\{\downarrow(\rho) \mid \rho \in cut(\Delta_G)\}$ (sentential form)
$s(G)$	137	Def 7.23	skeleton grammar
$P_=$	48	Def 2.62	pumping productions
$P_<$	48	Def 2.62	descending productions
$P_{<\omega}$	46	Def 2.56	bounded productions
P_ω	46	Def 2.56	unbounded productions
$\sup(\mathcal{X})$	21	Def 2.3	Supremum of a set of multisets.
$\sup(L)$	21	Def 2.5	Supremum of Parikh vectors
$\sup(L)(a)$	21	Def 2.5	a component of $\sup(L)$ (Parikh supremum)
$\sup(L)(\Sigma)$	21	Def 2.5	$\sum_{a\in\Sigma} \sup(L)(a)$ (Parikh supremum)
$\sup(G)$	46	Def 2.56	$\sup(cut(\Delta_G))$ (Parikh supremum)
g^{-1}	117	Def 6.13	pseudo inverse
$\mathcal{O}(f)$	58	Def 2.82	assymptotic notation
$\mathrm{o}(f)$	58	Def 2.82	assymptotic notation
$\Omega(f)$	58	Def 2.82	assymptotic notation
$\omega(f)$	58	Def 2.82	assymptotic notation
$\Theta(f)$	58	Def 2.82	assymptotic notation
$\mathcal{O}^T(f)$	58	Def 2.83	assymptotic notation
$\mathrm{o}^T(f)$	58	Def 2.82	assymptotic notation
$\Omega^T(f)$	58	Def 2.83	assymptotic notation
$\omega^T f$	58	Def 2.82	assymptotic notation
$\Theta^T(f)$	58	Def 2.83	assymptotic notation
$d_{L_\Delta}(i)$	60	Def 2.86	ambiguity
$\hat{d}_{L_\Delta}(n)$	61	Def 2.87	ambiguity
$U(L_\Delta)$	61	Def 2.88	ambiguity of tree language
d_G	61	Def 2.89	ambiguity
$D(\Gamma)$	70	Def 2.109	Dyck language with parenthesis of type Γ
$\mathcal{L}(R)$	110	Def 6.1	$\{u\#v^R \mid (u,v) \in R\}$
$\mathcal{R}(R)$	110	Def 6.1	$\{u^R\#v \mid (u,v) \in R\}$
$BMCFL$	104	Def 5.17	class of bounded marker languages
$UCFG$	62	Def 2.93	unambiguous grammars
$FCFG$	62	Def 2.93	grammars finitely ambiguous
$PCFG$	62	Def 2.93	grammars with polynomially bounded ambiguity
$ECFG$	62	Def 2.93	grammars with exponential ambiguity
$UCFL$	63	Def 2.95	unambiguous languages
$FCFL$	63	Def 2.95	languages with finite ambiguity
$PCFL$	63	Def 2.95	languages with polynomially bounded ambiguity

object	page	no	explanation
ECFL	63	Def 2.95	languages with exponential ambiguity
BDCFL	97	Sec 5.1.3	languages with bounded degree of direct ambiguity
mark	96	Def 5.4	
$predict\_prefix_G$	96	Def 5.5	
$im_G(n)$	96	Def 5.5	immediate ambiguity
$t_{\text{Earley}}(n)$	95	Sec 5.1.2	Zeitkomplexitt des Earleyalgorithmus.
$duplicate_G(n)$	95	Sec 5.1.2	

Index