

On Context-Free Grammar Induction by Incremental Compression

MSc Thesis (*Afstudeerscriptie*)

written by

Thomas Peetz

(born 23 January 1985 in Schweinfurt, Germany)

under the supervision of **Maarten van Someren** and **Pieter Adriaans**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam*.

Date of the public defense: **Members of the Thesis Committee:**
March 26, 2012

Maarten van Someren
Pieter Adriaans
Frank Veltman
Dick de Jongh
Christophe Florencio



INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Abstract State merging algorithms are predominant in Grammar Induction of finite state machines. This thesis extends the state merging approach to context-free grammars. It connects current standard implementations of state merging to compression-based learning. The exact same design principles are applied to the construction of a context-free grammar induction algorithm. The resulting induction algorithm is analyzed in terms of convergence, runtime, and data requirements.

To those who believe in happy endings

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Background and Related Work	1
1.3	Contributions	2
1.4	Structure of the Thesis	2
2	Automata Theory	3
2.1	Languages and Automata	3
2.2	Regular Languages: Finite State Automata	4
2.3	Context-Free Languages: Context-Free Grammars	5
2.4	Fragments of Recursion Theory	6
2.4.1	Turing Machines	6
2.4.2	Complexity Theory	7
2.4.3	Kolmogorov Complexity	8
3	Grammar Induction	9
3.1	A Sketch of Grammar Induction	9
3.2	Learnability of Languages	11
3.2.1	Identification in the Limit	12
3.2.2	Polynomial Runtime Complexity	13
3.2.3	Probably Approximately Correct Learning	13
3.3	Simplicity in Grammar Induction	14
3.3.1	The Minimum Description Length Principle	14
3.3.2	A Weak Form of Kolmogorov Complexity for Grammar Induction	17
3.3.3	A Computable Version of MDL for Grammar Induction	19
4	Incremental Compression Algorithms	21
4.1	State Merging Algorithms for FSAs	21
4.1.1	Initial Hypotheses	21
4.1.2	The Search Space of FSA Induction	22
4.1.3	Runtime and Heuristics	24
4.2	Iterative Compression Algorithms for CFGs	26
4.2.1	Size Measures Revisited	26
4.2.2	Initial Hypotheses	28
4.2.3	The Search Space of CFG Induction	29
4.2.4	Runtime and Heuristics	33
5	Properties and Applications of RuleSplit Grammars	34
5.1	Learning by Compression?	34
5.2	Applications of <i>LearnCFG</i>	35
5.3	Comparison to Other Algorithms	36
6	Conclusion	37

1 Introduction

1.1 Motivation

Grammar Induction is the task of learning a language from observations. Presented with a corpus of sentences, the learner must infer a grammar in order to generate new, previously unseen grammatical sentences. In this thesis, we focus on non-probabilistic approaches; the grammar need not reflect statistics about the corpus but only its *structure*. Many criteria can be applied to investigate whether the learner's strategy is *reasonable*: For instance, if it can be mathematically shown to work well for every possible set of observations, the learner can be said to solve the task. But even if such results cannot be established, the learner may still display good performance on real-world data sets.

For the simple class of regular languages, *state merging algorithms* have been examined thoroughly. Due to computational limitations, the best-performing state merging algorithms are forced to use heuristics which are known to often deviate from the correct grammar. Despite these problems, state merging algorithms have a very strong theoretical foundation and have long been shown to induce the correct grammars when sufficient computational resources and data are provided [dIH09].

Context-free grammars (CFGs) are slightly more complex than their regular counterparts. Many of the theoretical results concerning their learnability are negative, which may be one reason why research in appropriate learning algorithms has yet to find a similarly strong contestant for context-free grammar induction. The central idea of this thesis is to apply the design principles of state merging algorithms to CFGs, and to do so in a theoretically justified manner.

To do so, it must be understood that state merging algorithms perform an incremental *compression* of the sample sentences, converging towards a very compact grammar. While one would hope that the most compression also yields the correct grammar, this has been refuted for the general case [AV09]. Still, backed by empirical results, the idea of learning by compression lives on. Dubbed the *minimum description length* principle (MDL) it has become a branch of general learning theory in its own right. The second goal of this thesis is to investigate if and how MDL can be utilized for learning context-free grammars in this setting, and how close a state merging algorithm can get to learning by compression.

1.2 Background and Related Work

Being the simplest relevant class of languages, learnability results for regular languages have exhaustively been established [dIH09, DMV94], directly starting from what has become known as state merging algorithms [Lan92]. It was the winning entry of the Abbadingo grammar learning competition [LPP98] which combined the two most significant empirical results in the field of learning regular languages: The Blue-Fringe [JP98] strategy seeks to avoid exhaustive exploration of all possible grammars, while evidence-driven heuristics [dIH09] order candidate grammars by relevance.³

The theory of CFGs has been studied under a learning-theoretic perspective as well, albeit with

less positive results [dlHO04]. The Omphalos competition [SSCZ04], similar to Abbadingo, attempted to incite research for learning algorithms, and today the arguably most prominent approaches for CFG induction are EMILE [AV02a] and ADIOS [SHRE05]. While they are both based on merging rules, neither resembles state merging in the sense it has been used for regular languages: EMILE is not designed to explore the full range of possible candidate grammars, and ADIOS is a probabilistic graph algorithm. There exist approaches which are related to state merging techniques [CD10, NI00], but they rely on implicit assumptions and beyond empirical results, no justification has been brought forward.

If one wishes to provide such justification, some background in information-theoretic learning is necessary. The standard introduction to the field of MDL [LV93] also contains the material necessary to understand why learning by compression will not always work [AJ06, AV09] for grammar induction.

1.3 Contributions

This thesis extends the state merging approach to context-free grammars. It explicates the rationale behind the current standard implementations of state merging algorithms for regular languages, and proceeds to apply the exact same design principles to the construction of a CFG induction algorithm. Where there are a multitude of design choices, a mathematical analysis of the particular options is provided as to rid the induction algorithm of the trial-and-error flavor inherent to so many machine learning algorithms. The resulting induction algorithm is analyzed in terms of runtime and data requirements, and compared to other state-of-the-art CFG induction algorithms.

1.4 Structure of the Thesis

The remainder of this thesis is structured as follows. Section 2 provides the theoretical background in regular and context-free languages as well as tools to measure the complexity of computer programs. Section 3.1 will focus on Grammar Induction as such, introducing different notions of successful learning. Section 3.2 then analyzes the relationship between MDL and Grammar Induction, thereby describing what state merging algorithms are engineered to do from an MDL perspective. These considerations will be illustrated in section 4.1 with the case of regular language induction. The heart of this thesis can be found in section 4.2, where the extension to context-free languages happens. A CFG learning algorithm is derived from the principles of section 3.2, followed by an analysis of its runtime and data requirements. In section 5 the algorithm is fit into the bigger picture: It is examined whether it lives up to the expectations of MDL, and by comparing it to other CFG induction algorithms the central question is answered: If it is not learning the correct grammar, then what *is* it learning? Section 6 concludes.

2 Automata Theory

The objective of this thesis is the investigation of learnability of languages. In order to outline this vague goal more precisely, this chapter introduces the notion of a language in section 2.1 before moving on to some classes of processes which actually generate languages. The arguably simplest of such processes is the finite state automaton treated in section 2.2, but the focus of this thesis is on the context-free grammars described in section 2.3.¹ To allow for meaningful statements about programs, section 2.4.1 defines Turing machines, and sections 2.4.2 and 2.4.3 define measures for the complexity of a program based on requirements at runtime and on the program code as such, respectively.

2.1 Languages and Automata

As far as Grammar Induction is concerned, a language is nothing but a set of sequences of symbols. Such languages are deprived of any information beyond their structural features; in particular semantics. Yet their syntactic properties may well be the same as those found in actual, real-world sequential data such as natural language [CV07, NLHN09], genome information [Hea87, Sak05] or markup languages [Beh00, Fer01].

Let Σ be a non-empty finite set of symbols. This will be referred to as the *alphabet*; for simplicity it is generally assumed that $\Sigma = \{0, 1\}$. Let Σ^* be the set of all finite sequences that consist only of elements of Σ . Those will be called *sentences*. A set $L \subseteq \Sigma^*$ of sentences finally constitutes a *language*. A set \mathcal{L} of languages forms a *class* of languages.

Sentences will be denoted by $\mathbf{t} \equiv t_1 t_2 \dots t_{|\mathbf{t}|}$ where $|\mathbf{t}|$ is the *length* of \mathbf{t} , *i.e.*, the number of its symbols. To have a proper distinction from symbols we require that $|\mathbf{t}| \geq 2$ for any sentence \mathbf{t} . Akin to programming languages, $\mathbf{t}[i]$ and $\mathbf{t}[i:]$ denote $t_1 \dots t_i$ and $t_i \dots t_{|\mathbf{t}|}$, respectively. We write $\mathbf{u} \sqsubseteq \mathbf{t}$ if there are $i, j \in \mathbb{N}$ with $i < j$ such that $t_i \dots t_j \equiv \mathbf{u}$; in this case we call \mathbf{u} a *substring* of \mathbf{t} . We write $\sigma \in \mathbf{t}$ to indicate that \mathbf{t} *contains* a certain symbol $\sigma \in \Sigma$. A sentence of length zero, the *empty string*, is denoted λ . The *complement* of a language L is denoted \bar{L} .

Usually languages have names such as “English” or “XML 1.0”, however for the purpose of this thesis it is more convenient to use a different naming scheme. In Grammar Induction it is assumed that all languages stem from a particular process, so the exact specifications of such processes naturally are the most succinct naming scheme. In the following, a process generating a language will be referred to as an *automaton* and its specifications will be called a *grammar*. An automaton is a highly convenient way of specifying a language L : Instead of explicitly listing all of its sentences, we describe an automaton A which generates precisely L . Each automaton A represents exactly one language $L(A)$, and we will sometimes use the two interchangeably. In general, automata are simple computer programs, meaning they can be implemented on a Turing machine. The full class of such *computable* languages however is far too complex [Imm83].

Therefore, language classes are formed based on the type of automaton underlying the members of the class. As will be described shortly, different automata types have different expressive

¹Proofs of claims made in these sections can be found in any introductory textbook such as [HMU07].

power: The regularities underlying the language may be of different complexity. A preliminary hierarchy of automata classes was stipulated by Chomsky [Cho56]. While nowadays there are far more entries in this hierarchy, the main interest of Grammar Induction research has traditionally been focussed on the two simplest original language classes, finite state automata and pushdown automata [FB86a, FB86b]. These are also the main concern of this thesis.

2.2 Regular Languages: Finite State Automata

Finite State Automata (FSAs) and regular languages are *dual* in the sense that every FSA describes a regular language, and for every regular language there is an FSA implementing it.

Definition 2.1. The class of *regular languages* is the smallest class containing all

$$t ::= \sigma \mid \lambda \mid t_1 \cup t_2 \mid t_1 \circ t_2 \mid t^*,$$

where $\sigma \in \Sigma$, \circ denotes concatenation and $(\cdot)^*$ is the Kleene closure.

Since they are much easier to visualize it is convenient to think of a regular language in terms of an FSA.

Definition 2.2. A *finite state automaton* A is a quintuple $\langle \Sigma, S, s_1, F, \delta \rangle$ where Σ is an alphabet, S is a set of states, $s_1 \in S$ is the initial state, $F \subseteq S$ is a set of accepting states, and δ is a transition function.

As before, Σ is the alphabet recognized by A . The finite set S of states represent A 's limited capacity to memorize properties of a sentence during the testing process. In this sense s_1 contains absolutely no information, it is the initial state from which testing begins. The function $\delta : S \times \Sigma \rightarrow \mathcal{P}(S)$ describes transitions between states based on the next symbol of a sentence, where $\mathcal{P}(X)$ denotes the power set of X . We abbreviate the $|t|$ repeated state transitions $\delta(\dots \delta(s_1, t_1) \dots, t_{|t|})$ to $\delta^*(s_1, t)$. $F \subseteq S$ is the possibly empty set of accepting states: $t \in L(A)$ if and only if the testing of t leads A into a final state $s \in F$. If the automaton accepts every string in Σ^* it is called *universal*.

FSAs can be too complex to always deterministically switch between states based on one single symbol. An FSA is able to entertain different hypotheses on which structural property of the string it is currently observing. This is expressed in δ ranging over the power set of S , meaning that there may exist different paths through A , not all of which need to end up in an accepting state. In this case t is accepted if at least one accepting end state can be reached. Such FSAs are called *nondeterministic* finite state automata (NFA). If there is only one matching path for any sentence t through the automaton, i.e. the range of δ are singletons, one speaks of a *deterministic* finite state automaton (DFA). Aside from computational issues the distinction is usually not significant as every NFA can be turned into a DFA.

Definition 2.3. The *size* of FSA A is

$$M(A) = \sum_{s \in S} \sum_{\sigma \in \Sigma} |\delta(s, \sigma)| + 1.$$

One very particular approximation of M has traditionally received particular attention: Given an FSA A it is easy to build an *equivalent* FSA A' , meaning that $L(A) = L(A')$, with more states. The reverse clearly does not hold, at least not infinitely often.

Definition 2.4. An FSA A is *canonical* if there is no FSA A' such that $L(A) = L(A')$ and A' has less states than A .

While a unique canonical automaton always exists it is hard to find [Gol78] as well as to approximate [PW93].² By computing the canonical version of two FSAs it can be decided if they are equivalent.

2.3 Context-Free Languages: Context-Free Grammars

The class of automata most relevant for this thesis are *pushdown automata* (PDAs). For full generality, we work with nondeterministic PDAs; their dual is the class of *context-free* languages. Therefore, to avoid technical details PDAs will be defined by the grammar they represent [GPW82]. The class of context-free languages contains precisely the languages generated by a context-free grammar (CFG). It properly includes the regular languages [DPS02].

Definition 2.5. A *grammar* A is quadruple $\langle \Sigma, N, s, R \rangle$ where Σ is a finite set of terminals, N is a finite set of non-terminals, $s \in N$ is the start symbol, and R is a set of rewrite rules.

Σ again serves as the alphabet. The special nonterminal s is the start symbol which spans a sentence. The set $R \subseteq (N \cup \Sigma)^* \times (N \cup \Sigma)^*$ of rewrite rules determines how a given sequence of nonterminals can be expanded. Each rewrite rule takes the form $\alpha \Rightarrow \beta$ where the *head* α and the *body* β are any sequence of terminals and nonterminals. If $L(A) = \Sigma^*$ then A is called *universal*.

Definition 2.6. A *context-free* grammar is a grammar where all rewrite rules are of the form $n \Rightarrow \beta$ with $n \in N$.

To efficiently determine whether a string t is member of a language, the CYK [Coc69, You67, Kas65] algorithm can be used. Not unlike the case of FSAs it implements a nondeterministic way of checking certain structural properties of substrings of t , *i.e.*, it generates parse trees consistent with A over t . If at least one possible parse tree for t exists the membership query is answered positively. Deciding whether two given CFGs are equivalent is generally impossible, however.

Definition 2.7. The *size* of a CFG $G = \langle N, \Sigma, R, s \rangle$ is

$$M(G) = \sum_{n \Rightarrow \beta \in R} |\beta| + 1.$$

Other size measures have been used in the literature, but for reasons that will become obvious in section 2.4.3 we will work with M .

²More precisely, these problems are NP-complete (see Section 2.4.2).

Definition 2.8. Given a size measure M' , a CFG A is *canonical* if there is no equivalent CFG A' with $M(A') < M(A)$.

It should be noted that it is not decidable whether two given CFGs are equivalent.

2.4 Fragments of Recursion Theory

Recursion Theory [Coo02] is the branch of mathematics concerned with the automation of describable processes. Section 2.4.1 first shows how such a process can be represented by a program φ . Section 2.4.2 provides the tools to analyze the amount of computation required to execute φ , and section 2.4.3 shows how to measure φ 's inherent complexity.

2.4.1 Turing Machines

A Turing machine is a theoretical device used as the foundation of Recursion Theory. It can be described as a device running over an arbitrarily long (but finite) one-dimensional discretized tape of symbols, which is able to read and manipulate symbols on the tape based on an internal rule table.

Definition 2.9. A *Turing machine* φ is a quadruple $\langle \Sigma, S, s_1, \delta \rangle$ where Σ is an alphabet with $|\Sigma| \geq 2$, S is a finite set of states, s_1 is the start state, and δ is a transition function.

Σ once again is the alphabet φ works with. S is a set of states, of which s_1 is the initial state. Based on such states and the symbol currently read, φ can be programmed to perform actions. However, δ is more complex than in the previous cases.

During runtime of a program the *configuration* of the tape is in $(\Sigma^* \cup \{\lambda\}) \times \Sigma \times (\Sigma^* \cup \{\lambda\})$. The single symbol $\sigma \in \Sigma$ is the symbol currently observed by φ , with finite, possibly empty strings to the left and right. A *condition* of a Turing machine is an element of $S \times \Sigma$ consisting of φ 's state and the σ currently under inspection.

At any given time, φ is pointing at one field t on the tape. Let $I = \{l, r\} \cup \{p(\sigma) \mid \sigma \in \Sigma\}$ contain the instructions “move one field to the right”, “move one field to the left”, and “print a symbol $\sigma' \in \Sigma$ on t ”, respectively. Then $\delta \subseteq S \times \Sigma \times I \times S$ represents φ 's program as follows: If φ is in state s_i it will read the symbol σ occupying t . This condition may trigger one of the actions in I , which in turn will lead to a state change to $s_j \in S$. A Turing machine *halts* if it ends in a condition for which δ is undefined. While nondeterministic Turing machines are of great theoretical interest, in this thesis we only make use of deterministic Turing machines, *i.e.*, the range of δ is a set of singletons.

To treat Turing machines like the functions they implement, we write $\varphi(x)$ if the Turing machine φ receives x as *input*. It is provided to φ by the following convention: Initially φ is pointing to the first symbol of the input string which spans to the right of the Turing machine. The *output* of φ is the string left on the tape after φ has halted.

Proposition 2.10 ([Doy02]). *For any effectively calculable function f there is a Turing machine φ implementing f .*

This is known as the Church-Turing-Thesis. We will call a function f *computable* if there is a Turing machine implementing it, and if there is a Turing machine listing all of f 's range we call f *computably enumerable*. Otherwise f is *incomputable*. Unless indicated otherwise, all functions in this thesis are computable. We further call a class \mathcal{A} of well-defined objects *computable* if there is a Turing machine enumerating all $A_i \in \mathcal{A}$ in lexicographic order and increasing size, and *computably enumerable* if there is a Turing machine enumerating all members of \mathcal{A} . The class of all Turing machines is denoted Φ .

We have left the input and output of Turing machines somewhat unspecified. It is not clear how a given piece of information, be it a natural number or a structure like an automaton, should be translated to a string of symbols from Σ so that the Turing machine can work with it.

Definition 2.11. Given an alphabet Σ and a class \mathcal{A} of well-defined objects, an *encoding* $E_{\mathcal{A}} : \mathcal{A} \rightarrow \Sigma^*$ is a computable bijection. $E_{\mathcal{A}}(A)$ is a *code* for $A \in \mathcal{A}$.

The Church-Turing-Thesis guarantees that any mathematical object can effectively be handled by a Turing machine. In the case of computable classes, an encoding allows for outputting $\{E_{\mathcal{A}}(A) \mid A \in \mathcal{A}\}$ in lexicographic order, starting with the shortest elements. Clearly, definition 2.11 covers both FSAs and CFGs. For readability, we will often just write A for $E_{\mathcal{A}}(A)$. Throughout this thesis, for any \mathcal{A} one particular $E_{\mathcal{A}}$ will be used. In particular, there is one specific E_{Φ} . This leads to the particularly interesting corollary about the existence of universal Turing machines.

Definition 2.12. An *universal Turing machine* U is a Turing machine such that $U(\langle E_{\Phi}(\varphi), y \rangle) = \varphi(y)$.

That is, U takes the tuple $\langle E_{\Phi}(\varphi), y \rangle$ as input, and simulates the Turing machine φ on input y . There are multiple such U , and for the remainder of this thesis it is again imperative that U always denote the same universal Turing machine. If no additional input is provided we abbreviate $U(\langle E_{\Phi}(\varphi), \lambda \rangle) = U(E_{\Phi}(\varphi))$.

2.4.2 Complexity Theory

Complexity Theory [Pap94] is the analysis of computable functions in terms of Turing machines computing them. Given some function f , its central questions are: How many state transitions are at least needed for a Turing machine to compute f , and how long does the tape of any Turing machine computing f get? In this thesis we will only be concerned with the former question.

Definition 2.13. $f(x) = O(g(x))$ if there are real numbers $c, n, n_0 > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n > n_0$.

That is, $f(x) = O(g(x))$ if $g(x)$ dominates $f(x)$ by a multiplicative constant as x grows larger. We say that $f(x)$ is *polynomial in x* if $f(x) = O(p(x))$ for some polynomial p of x .

Definition 2.14. A Turing machine φ *runs in polynomial time* if for all x , the number of state transitions φ performs in computing $\varphi(x)$ is polynomial in x .

Polynomial-time Turing machines are generally considered feasible, while Turing machines with a runtime of $O(2^x)$ or greater are considered computationally hard. It has not been proven that this threshold is strict, *i.e.*, that there are Turing machines of runtime $O(2^x)$ which have no polynomial-time equivalent, but it is generally assumed that this is the case [For09].

Definition 2.15. A language L is *testable in polynomial time* if there exists a Turing machine φ_L such that $\varphi_L(t) = 1$ if $t \in L$ and 0 otherwise, and φ_L runs in polynomial time.

Both FSAs and CFGs are testable in polynomial time.

2.4.3 Kolmogorov Complexity

Universal Turing machines enable a comparison of program complexities. The *Kolmogorov complexity* $K(x)$ of a string x equals the smallest Turing machine to compute x . Intuitively K measures how much information is contained in x , which in turn may be a code for some structure like an FSA.

Definition 2.16 ([LV93]). The *conditional Kolmogorov complexity* of x given y is

$$K(x | y) = \min_{\varphi \in \Phi: U(\langle E_\Phi(\varphi), y \rangle) = x} |E_\Phi(\varphi)|.$$

Likewise we define the *unconditional Kolmogorov complexity* as $K(x) = \min_{\varphi \in \Phi: U(E_\Phi(\varphi)) = x} |E_\Phi(\varphi)|$.

There are two problematic aspects to Kolmogorov complexity. First, it is not absolute but always heavily depends on the particular choice of U and E_Φ . Therefore it is most meaningful *in the limit*, when $|x|$ goes to infinity. Will will reconsider this issue in section 3.3.2. The second problem is of a more fundamental nature.

Theorem 2.17 ([LV93]). K is *incomputable*.

$K(x)$ can however be approximated from above, by simulating all Turing machines simultaneously and keeping the shortest φ_i to output x and halt as current best approximation.

In order to put Kolmogorov complexity in a relation to the size measures introduced for automata, it is important to note that $K(x)$ is not monotonically increasing as $|x|$ grows: A simple string consisting of n zeroes is easier to describe than a random bit string of the same length. Size measures are proportional to the automaton's code length under a *simple* encoding, *i.e.*, one that does not exploit regularities in x to achieve shorter code lengths: We have both $M_{\text{FSA}} \propto E_{\text{FSA}}$ and $M_{\text{CFG}} \propto E_{\text{CFG}}$. This will be of importance in section 3.3.3.

3 Grammar Induction

Chapter 2 provided all the tools required to formally state the objective of this thesis. The fundamental questions tackled here are by no means new; their investigation has become known as Grammar Induction, a sketch of which is provided in section 3.1. This research field revolves around the learning of grammars, and it is imperative to define what counts as “learned”. Section 3.2 provides a literature overview over the different flavors thereof. A related idea, the minimum description length principle (MDL), provides the backbone for the work presented here. It receives special attention in section 3.3 where we derive MDL from general Kolmogorov complexity and fine-tune it for our needs.

3.1 A Sketch of Grammar Induction

Grammar Induction, sometimes also called Grammar Inference, is a subfield of Computational Linguistics [Kor08] which in turn is concerned with the development of computer programs that are able to handle natural language. Even for the syntactical level, this optimistic goal is however far from achieved [GAJT07, Zho07]. Most current research is taking one of two main directions.

Language Modeling [JM08] works under the assumption that there is a probabilistic process P from which a language L is sampled. The hope is that if one has *positive training data* $I_+ \subseteq L$ with all properties sufficiently similar to L , it should be possible to estimate P from I_+ , *i.e.*, to reverse-engineer P in order to have a general model describing L . The quest for P starts out by selecting an *estimator* R_M which is run on I_+ . Precisely modeling and thereby *overfitting* I_+ , it returns a member $R_M(I_+)$ of a class of models \mathcal{M} .

Individual estimators may be problematic for a number of reasons. First and foremost they may be computationally costly [Sim02] and thus difficult to use if I_+ is big. Next, estimators may be so successful in overfitting I_+ that $R_M(I_+)$ has little to say about $L \setminus I_+$. Such cases can be tackled by *smoothing* [CG96]. Another, deeper issue arises when an estimator turns out to be *inconsistent* with the statistics of the data [Joh98] despite yielding good results in experiments. In such cases even $R_M(L)$ will not correctly describe L . The same problem occurs if \mathcal{M} is badly chosen and does not even contain P . Finally there is the fundamental problem that the main hypothesis is necessarily violated: Non-artificial training sets are biased towards true statements and the speakers’ culture. False but perfectly grammatical sentences are significantly underrepresented in I_+ .

It seems fair to say that the question if $R_M(I_+)$ could do L justice at all is often deliberately ignored. Given the omnipresent evidence that natural languages are in fact learnable to a near-perfect degree at least by biological devices, Language Modeling leaves the question of perfect learnability aside and focuses on robustly dealing with as big a subset of L as possible.

Language Modeling generally directly computes one single model. In this sense, it is a top-down approach to the processing of natural language. *Grammar Induction* [dlH10] is its bottom-up counterpart. Instead of immediately focussing on natural language in all its problematic richness, it sets intermediate steps by grouping languages into classes based on severely limited processes,

thus shifting the focus from natural to artificial languages generated by automata. These limitations usually constrain the complexity and expressiveness of the language L in question; they also imply that the *inference algorithm* $R_{\mathcal{A}}$ must return a specific automaton $A \in \mathcal{A}$, with \mathcal{A} being the automata class in question.

To further ease the difficulties of the induction task the assumption that the artificial language L is generated by a probabilistic process is often dropped, making the learning task *distribution-free*. Therefore on first sight, the main questions Grammar Induction seems to ask are the following:

Let the class \mathcal{A} of automata be dual to the class \mathcal{L} of languages. Given an arbitrary $L \in \mathcal{L}$ and training data $I_+ \subseteq L$, is there an inference algorithm $R_{\mathcal{A}}$ such that $R_{\mathcal{A}}(I_+)$ precisely describes L ? And if such an algorithm exists, does it run in a reasonable amount of time?

Already this fundamental question is based on the bold assumption that some form of inference is actually necessary. Without any information beyond I_+ there is no immediate reason to assume that I_+ is not the whole language, in which case I_+ is the best specification there is. Likewise, even if it is known that $I_+ \subset L$ (*i.e.*, I_+ is a proper subset of L) any guess about L is moot, for even $L = \Sigma^*$ is consistent with the given data. A remedy is to provide a set I_- of *negative training data* such that $I_- \cap L = \emptyset$, establishing a more well-defined learning problem by ruling out universal grammars as valid solutions.

This is precisely the scenario this thesis is concerned with: Training data $D = \langle I_+, I_- \rangle$ is fixed, finite and available as a bulk. It is known that $I_+ \subseteq L \subseteq \Sigma^*$ and assumed that L is infinite.³ This implies $I_+ \subset L$ and introduces the need for inference. It does however say nothing about the shape of L or how representative D is in that respect, even though it is known that I_+ and L are generated by the very same process.

In the Language Modeling setting it is assumed that both I_+ and L are distributed according to the probabilistic process P , and I_+ is merely a sparser version of L . Therefore a large D is likely to contain enough relevant information to yield an accurate $R_{\mathcal{M}}(I_+)$. In distribution-free Grammar Induction this assumption cannot be made and I_+ may well be generable by a much simpler automaton than L while I_- is not guaranteed to be very revealing either. One must establish which properties the data D must exhibit so that $R_{\mathcal{A}}(D)$ describes L , which can only be answered if $R_{\mathcal{A}}$ is known. We get the following reformulation of the main question of Grammar Induction:

Let the class \mathcal{A} of automata be dual to the class \mathcal{L} of languages. Is there an algorithm $R_{\mathcal{A}}$ such that for arbitrary $L \in \mathcal{L}$, there is a set $D = \langle I_+, I_- \rangle$ with $I_+ \subset L$ and $I_- \cap L = \emptyset$ for which $R_{\mathcal{A}}(D)$ precisely describes L ? If so, what are the properties D must exhibit for this to hold, and is $R_{\mathcal{A}}$ computable in a reasonable amount of time for such D ?

Grammar Induction has produced many important results for a large variety of language classes. A compact, technical introduction can be found in [dlH05] or the more specialized [dlH09]

³Due to their lack of flexibility, finite languages possess severely limited expressive power and are generally dismissed as not very interesting.

and [dlHO04] for FSA and CFG induction, respectively. [CKP03] puts greater emphasis on algorithms and implementations. To better understand the individual contributions of Grammar Induction research, we will now specify the different learning paradigms met in the field.

3.2 Learnability of Languages

With the aim of Grammar Induction specified, the next step is to fix the conditions under which the learning takes place. It is clear that the learner in this setting is a computable algorithm R of some kind. Yet, as it stands, this task is vastly underspecified unless one agrees on a number of details. Following [Gol67], the following must be established in order to establish a common ground:

- a way to refer to individual languages,
- a method how information is presented to the learner, and
- a definition of learnability.

The first item has already be defined; throughout this thesis grammars will be used as names for languages. Concerning the second item, the notion of training data D has been introduced in section 3.1. It contains both positive and negative data, always correctly labeled. Furthermore it is fixed and immediately available, in particular it is not a steady stream of new data. It is not possible for the learner to make inquiries about unseen data, *e.g.*, to ask whether $t \in L$ for a sentence $t \notin D$.⁴ It is hence reasonable to define the learner as a function $R_{\mathcal{A}} : \mathcal{P}(\Sigma^*) \times \mathcal{P}(\Sigma^*) \rightarrow \mathcal{A}$.

It remains to define when a learner has successfully learned L , and thus to find a criterion to make statements about different learners' quality. In Language Modeling it is customary to withhold a small amount of data $D_T \subset D$, the *test set*, and the ability of $R_{\mathcal{M}}(D \setminus D_T)$ to *generalize* over this "unseen" D_T is used to establish a *preorder* $<_{D_T}$ over the union $\bigcup_{i \in I} \mathcal{H}_i$ of possible outputs (*hypotheses*) of learners $R_{\mathcal{M}_i}$. In particular, $H_i <_{D_T} H_j$ (H_j is *more likely correct* than H_i) if H_j is agreeing with D_T more than H_i is. Accordingly, R_j is deemed better than R_i if $R_i(I_+) < R_j(I_+)$. Such rankings are of great importance for probabilistic settings, where the correct parameters are likely not exactly determinable due to sparseness.

In principle $<_{D_T}$ can be used for Grammar Induction as well, but the less empirical flavor of this field suggests a different approach. In Language Modeling a learner must construct a grammar and estimate the parameters of P . In distribution-free Grammar Induction the estimation is not required, and one only demands that $R_{\mathcal{A}}(D)$ describes the target language. As any of the infinitely many grammars describing L is accepted as a success, this allows for a greater focus on efficient algorithms.

Without establishing an elaborate preorder over all possible outputs, Grammar Induction strictly defines what counts as successful learning. The requirement that $R_{\mathcal{A}}(D)$ describes L is invariant through all notions of learnability, but various definitions put additional demands on the learner. Sections 3.2.1 through 3.2.3 provide an overview of the most common such demands.

⁴We use $d \in D$ in a suggestive way instead of $D = \langle I_+, I_- \rangle, d \in I_+ \cup I_-$.

3.2.1 Identification in the Limit

The historically first definition of learnability was provided by [Gol67]. It is also the most relaxed and impractical such definition, not demanding any bounds.

Definition 3.1. Given a language L , a learner $R_{\mathcal{A}}$ identifies L in the limit if for some representation e there is a $J_+ = \langle e(j_1^+), \dots, e(j_n^+) \rangle$ with $j_i^+ \in L$ and a $J_- = \langle e(j_1^-), \dots, e(j_m^-) \rangle$ with $j_i^- \notin L$ such that for all $J'_+ = \langle e(j_1^+), \dots, e(j_n^+), \dots, e(j_{n'}^+) \rangle$ with $j_i^+ \in L$ and a $J'_- = \langle e(j_1^-), \dots, e(j_m^-), \dots, e(j_{m'}^-) \rangle$ with $j_i^- \notin L$ it holds that $R_{\mathcal{A}}(\langle J_+, J_- \rangle) = R_{\mathcal{A}}(\langle J'_+, J'_- \rangle) = A$ and $L(A) = L$.

A class \mathcal{L} of languages is *identifiable in the limit* if there is a learner that identifies every $L \in \mathcal{L}$ in the limit.

Identification in the Limit (IITL) is best understood as learning from a stream of positive and negative data: At some point $R_{\mathcal{A}}$ reaches a *fixpoint* after which new data will not influence its output anymore. A crucial detail is that the learner is not required to be aware of its success. If this awareness were required almost no interesting class of languages could be learned [Gol67] because there are scenarios in which a learner changes its hypothesis arbitrarily often [Ang80, AM10].

A class \mathcal{L} of languages is called *superfinite* if \mathcal{L} contains all finite languages and at least one infinite language. The following result signifies the importance of negative data for Grammar Induction, if one takes for granted that any interesting language class contains all finite languages.

Theorem 3.2 ([Gol67]). *Any superfinite class of languages is not identifiable in the limit from positive data alone.*

This is related to the discussion about the need to generalize in section 3.1. If \mathcal{L} contains all finite languages then $R_{\mathcal{A}}$ can at best assume that I_+ describes the target language L exhaustively, and $R_{\mathcal{A}}(I_+)$ accepts precisely I_+ . This way the infinite language can never be identified. Likewise, all finite languages *can* be identified. However, if there is negative data available, the picture changes.

Theorem 3.3 ([Gol67]). *Any recursively enumerable class of languages can be identified in the limit using positive and negative examples.*

The proof is not very involved, it simply has the learner enumerate all grammars and choose the first one consistent with the training data. Its implications however are deep: FSAs and CFGs are identifiable in this scenario.

If a class of languages is IITL this has little immediate bearing on most practical applications. IITL has nothing to say about the runtime of a learner $R_{\mathcal{A}}$, about the amount of training data needed or even about the behavior of $R_{\mathcal{A}}$ before the fixpoint is reached: Ideally, more training data should lead to an annealing to the correct solution. These issues are ignored by IITL. In response to this other, more practical definitions of learnability have been established, covering these points in turn.

3.2.2 Polynomial Runtime Complexity

In times of Moore's Law [Sch97] it is not easy to define the intuitively understood term *reasonable runtime*. A generally accepted criterion for an algorithm to be practical is that it finishes in *polynomial time*, but in the case of Grammar Induction is not clear in which variables the runtime should be polynomial. Skipping exhaustive discussion of the parameters [Pit89], we provide the following definitions.

We say that a learner R makes a *prediction error* if its hypothesis at step i is not consistent with the $i + 1$ st example it observes. The runtime of R updating the current hypothesis is called its *update time*.

Definition 3.4 ([Pit89]). Given a language L and training data $\langle I_+, I_- \rangle$ with $I_+ \subset L$ and $I_- \cap L = \emptyset$, a learner $R_{\mathcal{A}}$ *identifies L in polynomial time* if $R_{\mathcal{A}}$

- identifies L in the limit, and
- the number of prediction errors is polynomial in the target automaton's size n , and
- $R_{\mathcal{A}}$ has update time polynomial in n and the sum of lengths of the data samples d_i observed so far.

A class \mathcal{L} of languages is *identifiable in polynomial time* if there is a learner that identifies every $L \in \mathcal{L}$ in polynomial time.

A caveat is that [Pit89] does not actually define a size measure; rather it requires that the size measure be *reasonable*. The definitions provided in sections 2.2 and 2.3 certainly fall into this cluster.

It turns out that neither FSAs nor CFGs can be learned under this definition [dlH95]. The proof involves the construction of two different languages that can only be separated by a string larger than 2^n . As there is no method to test for equality of two FSAs or CFGs that runs in polynomial time, the result follows. Grammar Induction for these classes is hence considered impractical, which reiterates the need of precise runtime analysis of induction algorithms.

3.2.3 Probably Approximately Correct Learning

Technical subtleties aside, identifiability in polynomial time seems an interesting property for most real-life learning tasks. However, if the target language \mathcal{L} is not identifiable in the limit, polynomial time identification fails to provide information beyond this fact. Even if no supremum exists, the order $H_i < H_j < H_k < \dots$ might be enumerable. Doing so is the idea underlying *probably approximately correct* (PAC) learning.

Since almost all PAC-results concerning CFGs are negative [dlHO04] it is not necessary to dive into the technical details here. Informally, a class of languages is PAC-learnable if there is an algorithm which, with arbitrarily high probability, generalizes arbitrarily well with enough training data [Val84]. The key element is that when the learner is presented more data, the probability of prediction errors decreases. While PAC-learning does not require absolute convergence,

it demands that the learner be able to get arbitrarily close to the best solution available. PAC-learnability thus incorporates two crucial features: It requires that the learner be fast and only sufficiently correct, for all possible data within its domain. This is akin to real world learning problems: Biological devices do not learn everything exactly; rather they learn quickly at the cost of occasional errors.

While Grammar Induction is distribution-free, the requirement of better results given more data is still relevant. Firstly, greedy best-first approaches are much easier to implement than strategies to exhaust the entire hypothesis space. Secondly, when multiple hypotheses are possible for the next iteration of an algorithm, it is good to have a strategy of picking the optimal one — or a proof that such a strategy is not needed.

As noted, PAC-learnability is independent of the data presented. If one assumes that the data is chosen in a benign way, much more positive results can be obtained for CFG induction [LV91]. This paradigm, called *simple PAC-learnability*, has been investigated in a series of papers [DDG96, PH97, Adr99], which branches away from the goals of this thesis.

This section has formalized the usual requirements of a learning algorithm, such as that it be correct and efficient. The following section will illuminate whether compression-based learning can deliver these properties.

3.3 Simplicity in Grammar Induction

3.3.1 The Minimum Description Length Principle

The definitions of learnability in section 3.2 are entirely indifferent about $R_{\mathcal{A}}(D)$ as long as it describes L . It is however easy to see that different grammars can well describe the same language, and it is not obvious why there should not be a way to tell which grammar is *better* for any meaning of this term, such as smaller, less complex or more interesting. This section is dedicated to re-introducing a preorder on different but equivalent automata.

The Minimum Description Length principle (MDL) does not occur in the list of definitions for successful learning because it shifts the attention from properties of the learner to its output. MDL is a direct implementation of Occam’s Razor, made mathematically precise by means of Kolmogorov complexity (see section 2.4.3). Occam’s Razor is commonly summarized as “Concepts must not be multiplied beyond necessity” [Bri10], which in the context of inductive inference translates to:

If multiple hypotheses perform equally well, the simplest one is most likely correct.

The words “perform” and “simple” must be specified more precisely.

In every language learning task there is a class \mathcal{H} of hypotheses, usually grammars, which may or may not explain for the training data D . The first task for a learner $R_{\mathcal{H}}$ obviously is to single out the set $\mathcal{H}_D \subseteq \mathcal{H}$ of hypotheses which are *consistent* with D . Generally \mathcal{H}_D is relatively big because of the sparseness of D , but the inference algorithm should decide on one single $\hat{H} \in \mathcal{H}_D$ which is *optimal* in \mathcal{H} wrt. D .

Section 3.2 described how an likelihood order $<_{D_T}$ over different hypotheses can be established by comparing them to previously withheld data D_T . While this conceptually remains a valid method in the realm of MDL [Grü05], there is another method which does not rely on empirical evaluation of a hypothesis. If two hypotheses H_1, H_2 both are consistent with an observation D , MDL claims that H_2 is more likely correct than H_1 if it is simpler, that is, $H_1 <_K H_2$ if $K(H_2) < K(H_1)$ where $K(x)$ is the Kolmogorov complexity of x . This makes $\hat{H} = \arg \min_{H \in \mathcal{H}_D} K(H)$ most likely correct hypothesis.

This definition may seem ad-hoc, but it can easily be motivated. We will do so starting from basic statistics. There, a *prior* probability distribution P over \mathcal{H} expresses the initial belief about the probability of any given hypothesis. Any observation $d \in D$ leads to a *belief update*, *i.e.*, a revision of the hypotheses' likelihood based on d . Since we are working in a distribution-free setting, $P(D | H)$ is binary: $P(\langle I_+, I_- \rangle | H) = 1$ if and only if $t \in I_+ \rightarrow t \in L(H)$ and $t \in I_- \rightarrow t \notin L(H)$ for all t . Otherwise, $P(D | H) = 0$. After all of D has been observed, the *maximum a posteriori* hypothesis \hat{H} is chosen as to maximize its likelihood given D . That is, $\hat{H} = \arg \max_{H \in \mathcal{H}} P(H | D)$.

Using Bayes' formula this can be reformulated as follows:

$$\begin{aligned} \hat{H} &= \arg \max_{H \in \mathcal{H}} P(H | D) \\ &= \arg \max_{H \in \mathcal{H}} \frac{P(H) \cdot P(D | H)}{P(D)}. \end{aligned}$$

In the setting of language learning, the observation D is a given, so $P(D) = 1$. If we can furthermore *a priori* restrict \mathcal{H} to the class \mathcal{H}_D of hypotheses consistent with D , then clearly $P(D | H) = 1$ as well and $\arg \max_{H \in \mathcal{H}_D} P(H)$ only depends on the prior.

Definition 3.5 ([Sol64]). The *universal a priori probability* is

$$\hat{P}(x) = \sum_{\varphi \in \Phi: U(\varphi)=x} 2^{-E_\Phi(\varphi)},$$

where U is the reference universal Turing machine from Definition 2.12.

So the prior probability of a string x is the probability of U generating x from random bit strings as input.

Theorem 3.6 ([LV93]). *There is a constant c such that $\hat{P}(x) = 2^{-K(x)} \cdot c$ for all x .*

This central theorem replaces the cumbersome \hat{P} with the simpler concept of Kolmogorov complexity: It shows that one can substitute $2^{-K(x)}$ for $\hat{P}(x)$ with only marginal error. It also reflects MDL's general bias to favor shorter codes exponentially, which is in accordance with Occam's Razor. It remains to show why this specific prior probability is suitable for inductive inference.

Since it is known that $I_+ \subset L$ it is clear from the start that a hypothesis which does not *generalize* will not be ranked high. The main difference between the evaluation-based approach and the Kolmogorov complexity-driven version of hypothesis selection is the unpredictable bias of

the former: Any algorithm designed with an empirical mindset will generalize in some way, but there are no constraints on how to achieve this. Even extending I_+ with random sentences can be successful in many cases [WM97], or the mechanisms leading to generalization may only be “intuitively” plausible or reasonable. This is less a problem of the learning algorithms themselves but one of their design principles, and accepting an objective method to induce generalization would quickly remedy this issue.

Given a set D of observations, Grammar Induction asks for an explanation for D . In principle, any model accounting for all $d_i \in D$ might do, but some models will be very complex and fine-tuned to precisely cover D while others will be of simpler structure but lump D together with other, potentially unrelated data.

It is difficult to balance the over- and underfitting of D . One severe restriction on underfitting is already implemented by confining the hypothesis space of a learner to a certain automata class. Beyond this, the only property of data MDL is capable of expressing is complexity relative to a model.

Definition 3.7 ([AV09]). Let A, H with $A \subseteq H$ be sets of cardinality a and h , respectively. The *randomness deficiency* of A wrt. H is

$$\Delta(A | H) = \left| E_{\mathbb{N}} \left(\binom{h}{a} \right) \right| - K(A | H).$$

The binomial coefficient serves as an upper bound for $K(A | H)$: The easiest way to encode an element $h \in H$ given H itself is by reference. Put all $h_i \in H$ in lexicographic order, then describe h by its list index. Doing this for all of $h \in A$ provides an upper bound for $K(A | H)$. A small randomness deficiency means that A is very *typical* for A , in the sense that it has no special structure that is simpler than other equally large subsets of H ; we say that it is *random in H* . If A has high randomness deficiency there are better descriptions than the above which do not make use of H . In this case, A is not very typical.⁵

Therefore the task of model selection is to select the $\hat{H} \in \mathcal{H}$ such that $\Delta(D | \hat{H})$ is minimal. If \hat{H} can be found, it guarantees that most $h \in \hat{H}$ are similarly random to elements of A . Thus, finding \hat{H} does indeed lead to generalization.

It must be mentioned that all of the above assumes that $A \subseteq H$ and that \hat{H} does indeed exist (it need not be unique). In the setting of Grammar Induction all of this is fulfilled, as discussed earlier. The next section will investigate how the idea of MDL can actually be applied to Grammar Induction.

⁵For simplicity, we explained randomness deficiency in relation to finite sets only. To see that randomness deficiency is similarly well-defined for infinite sets, recall the *benign* distributions of section 3.2.3. Such distributions are very close to the universal a priori distribution which favors samples of low complexity. For infinite sets, the randomness deficiency of definition 3.7 must be replaced by a distance measure from the universal distribution. Let the data D be described by a probability distribution P_D . The *Kullback-Leibler-divergence* between P_D and the universal probability of D is a measure [ES03] which coincides with the coding-theoretic interpretation of randomness deficiency [AASV09]. A formal derivation requires some decisions on how to construct P_D from a data set however, and has thus been precluded from the formal part of this thesis.

3.3.2 A Weak Form of Kolmogorov Complexity for Grammar Induction

This section will provide the formal model in which MDL can be applied to Grammar Induction. The goal is to allow for statements about the complexity of a grammar A , isolated from actual programs implementing A . The foundation of this undertaking is a universal Turing machine U which can be programmed with a Turing program φ and henceforth implements it. In the following we omit the mention of U and talk about φ only.

It is easy to believe that for each class \mathcal{A} of automata below Turing complexity there is a Turing program $\varphi_{\mathcal{A}}^{(E_{\mathcal{A}})}$ which takes a representation of a grammar $A \in \mathcal{A}$ as input and simulates the automaton corresponding to A . In order to do so, however, A must be encoded by an encoding $E_{\mathcal{A}}$. As mentioned earlier, for any class \mathcal{A} we work with one fixed $E_{\mathcal{A}}$. We therefore abbreviate $\varphi_{\mathcal{A}} = \varphi_{\mathcal{A}}^{(E_{\mathcal{A}})}$. This $\varphi_{\mathcal{A}}$ is best understood as a *blank automaton* of type \mathcal{A} that has yet to receive a grammar, but can only understand codes of $E_{\mathcal{A}}$. We further abbreviate $\varphi_A = \varphi_{\mathcal{A}}(E_{\mathcal{A}}(A))$.

Once $\varphi_{\mathcal{A}}$ has been instantiated with a grammar A it acts as a function $\varphi_A : \Sigma^* \rightarrow \{0, 1\}$ such that $\varphi_A(t) = 1$ iff $t \in L(A)$. This φ_A is total since unlike for general Turing programs, for automata below Turing complexity the question whether $t \in L(A)$ is decidable for all A and t .

Definition 3.8. The *modular code length* of a Turing machine φ_A implementing a grammar $A \in \mathcal{A}$ is

$$K_{mod}(\varphi_A) = K(\varphi_{\mathcal{A}}) + |E_{\mathcal{A}}(A)|.$$

The modular code length constitutes a first, intuitive attempt to define the complexity of A relative to the class \mathcal{A} : It has a clear separation of the grammar A from the rest and all programs are restricted to deal with the class \mathcal{A} . Like Kolmogorov complexity it is invariant under different encodings $E'_{\mathcal{A}}$ up to an additive constant which depends on $E_{\mathcal{A}}$ and $E'_{\mathcal{A}}$. By exposing its degrees of freedom we shall now see that definition 3.8 is unsatisfactory for two reasons.

Pitfall 1: Prior Knowledge Recall the convention that for every \mathcal{A} there is only one $E_{\mathcal{A}}$, which in turn $\varphi_{\mathcal{A}}$ depends on: As of now, no further constraints have been placed on $E_{\mathcal{A}}$ and $\varphi_{\mathcal{A}}$. This may lead to an unwanted side effect when the grammar to encode is known in advance.

Theorem 3.9. For all $A \in \mathcal{A}$ there is an encoding $E_{\mathcal{A}}$ and a blank \mathcal{A} -automaton $\varphi_{\mathcal{A}}$ such that $|E_{\mathcal{A}}(A)| = 0$ in the modular code of φ_A .

Proof. Given A , design an encoding $E_{\mathcal{A}}$ such that $E_{\mathcal{A}}(A) = \lambda$ and write a Turing program $\varphi_{\mathcal{A}}$ which, when getting the empty string as input, outputs the program for φ_A . Clearly $|E_{\mathcal{A}}(A)| = 0$. \square

It is hence possible to assign very short code lengths to arbitrarily complex grammars. Theorem 3.9 covers the extreme case where one single grammar is hard-coded in $\varphi_{\mathcal{A}}$. This can be extended to arbitrarily many grammars: The larger the code for $\varphi_{\mathcal{A}}$, the more grammars it can store explicitly. Their output could then be triggered by very short inputs mimicking codes: A binary string of length n can invoke up to 2^n grammars. In order to do so however, these grammars and

their codes must both be stored explicitly in $\varphi_{\mathcal{A}}$. This is only possible if the critical grammars are known in advance. The obvious solution to this “cheating” problem chooses $\varphi_{\mathcal{A}}$ as simple as possible, thereby limiting its capacity to store grammars. We will denote this minimal blank \mathcal{A} -automaton $\hat{\varphi}_{\mathcal{A}}$.⁶

It must be noted that the problem of prior knowledge does not cover all cases where more complex grammars get shorter codes than simpler ones. Using $\hat{\varphi}$ therefore does not establish that $K^{mod}(\varphi_A) \propto K(\varphi_A)$.

Pitfall 2: Grammars vs. Programs If we replace the blank \mathcal{A} -automaton in definition 3.8 with $\hat{\varphi}_{\mathcal{A}}$, then $K_{mod}(\varphi_A)$ largely depends on $|E_{\mathcal{A}}(A)|$: $\hat{\varphi}_{\mathcal{A}}$ is independent of A by definition, and there is no need to store A in $E_{\mathcal{A}}$ since it would only be needed if A should receive a specific code. Therefore $K_{mod}(\varphi_A) = |E_{\mathcal{A}}(A)| + \mathcal{O}(1)$.

Theorem 3.10. *For all $A \in \mathcal{A}$ there are encodings $E_{\mathcal{A}}$ and blank \mathcal{A} -automata $\varphi_{\mathcal{A}}$ such that $K_{mod}(\varphi_A) = K(E_{\mathcal{A}}(A))$.*

Proof. Given A , write a Turing program $\hat{E}_{\mathcal{A}}$ which takes A as input and directly outputs the program for φ_A . Such a program satisfies all properties of an encoding. Then $K(\hat{E}_{\mathcal{A}}(A)) = K(\varphi_A)$ and $\varphi_{\mathcal{A}}$ is the identity function, so $K(\varphi_{\mathcal{A}}) = 0$. \square

Theorem 3.10 shows that $\varphi_{\mathcal{A}}$ can be incorporated into the encoding, eliminating the need for an extra blank \mathcal{A} -automaton. Using such a degenerate encoding is problematic for a number of reasons. Firstly it is precisely the merging of components of the modular code length that we are trying to avoid. Secondly, all algorithms for Grammar Induction must return a grammar, and it is not necessarily possible to extract the grammar from this complex program: Recall that it is not for all automata classes below Turing complexity possible to decide if two automata describe the same language. Finally, \hat{E} relies on the full Kolmogorov complexity and is therefore incomputable. In general, it is not very intuitive that $E(A)$ should be a program: Every computational aspect one might want in E can be outsourced to the interpreting $\varphi_{\mathcal{A}}$. Therefore, $E_{\mathcal{A}}(A)$ cannot be a program and the separate $\varphi_{\mathcal{A}}$ cannot be avoided.

Definition 3.11. Given a class \mathcal{A} of automata and a corresponding encoding $E_{\mathcal{A}}$, the \mathcal{A} -complexity of a grammar $A \in \mathcal{A}$ is

$$K_{\mathcal{A}}(A) = \min_{A \in \mathcal{A}: \hat{\varphi}_{\mathcal{A}}(A) = \varphi_A} |E_{\mathcal{A}}(A)|.$$

The \mathcal{A} -complexity measures the complexity of an automaton $A \in \mathcal{A}$ relative to all other automata in \mathcal{A} , under a fixed representation. The \mathcal{A} -complexity is a restricted version of Kolmogorov complexity in that it prevents switching between different models: Even a simple CFG which can be described by an FSA will always be described relative to the class of CFGs. Therefore:

Theorem 3.12. $K_{\mathcal{A}}(A) = |E_{\mathcal{A}}(A)| + \mathcal{O}(1)$ for all $A \in \mathcal{A}$.

⁶While it cannot be determined which $\varphi_{\mathcal{A}}$ has minimal Kolmogorov complexity, for the remainder of this thesis it is completely sufficient to take any non-cheating such program. The same clearly holds, mutatis mutandis, for U .

Due to the general undecidability of two automata's equivalence the \mathcal{A} -complexity still cannot be directly computed. Nevertheless it is a robust incentive for enumerating grammars with decreasing complexity: Theorem 3.10 shows that for Grammar Induction the grammar must actually be separated from the interpreting function. The following section investigates the feasibility of compressing a grammar.

3.3.3 A Computable Version of MDL for Grammar Induction

This section will illustrate considerations of sections 3.3.1 and 3.3.2 by describing how MDL can be used for Grammar Induction.

Definition 3.13. Two total functions f, g are *functionally equivalent* wrt. a finite set D if for all $d \in D$ we have $f(d) = g(d)$.

While the equivalence of two CFGs (or more expressive automata) is not decidable, functional equivalence wrt. finite D can be easily tested for. We now go back to the learning problem outlined in section 3.1. Assume some hypothesis $H \in \mathcal{H}_D$, the space of hypotheses consistent with D , can be found. H may then return arbitrary values for sentences not in D .

An *incremental compression algorithm* for hypotheses is a computable process $Q : \mathcal{H}_D \rightarrow \mathcal{H}_D$ such that A and $Q(A)$ are functionally equivalent wrt. D and $K_{\mathcal{A}}(Q(A)) \leq K_{\mathcal{A}}(A)$. In order to be of practical significance, for any input A there should be a fixpoint such that $Q(A) = A$ where A is of minimal \mathcal{A} -complexity. Furthermore, Q 's runtime requirements should be both determinable and reasonable. Then we get the following proposition, which is the MDL approach to Grammar Induction.

Proposition 3.14. *Given training data D , and a hypothesis $H \in \mathcal{H}$ consistent with D , any iterative compression algorithm $R = Q^*$ minimizing $K_{\mathcal{H}}(H)$ will solve the grammar induction problem.*

However, there is more: The \mathcal{A} -complexity in full generality is incomputable, thus a *simple* restricted encoding must be used for hypotheses in order to compare the complexity (*i.e.*, the code length) of different hypotheses. In this context, simple encodings perform no additional inference; the encoding is a very straightforward process of low Kolmogorov complexity. In sections 2.2 and 2.3 we defined size measures M : If the automaton A becomes larger, so does $M(A)$. This is not an overly strong constraint, and in fact all that is required is that the size measure be proportional to simple encodings.

That is not to say that the results of Q are independent of the chosen size measure. For example, let $M_1(A)$ emphasize the number of rewrite rules of a CFG A , and let $M_2(A)$ utilize an Huffman coding [Huf07] of A 's rule set: Then it is not at all obvious how an algorithm R effectively minimizing $M_1(A)$ will behave in terms of $M_2(R(A))$, and vice versa.

This point is worth reiterating. It is easy to see that under a Turing-complete compression mechanism, the code length $|E(x)|$ of some x is invariant under different objective encodings E_1, E_2 up to a constant $c(E_1, E_2)$. This constant is the amount of code it takes to decode $E_1(x)$

and re-encode it to $E_2(x)$. As soon as the difference between $|E_1(x)|$ and $|E_2(x)|$ outgrows the constant $c(E_1, E_2)$ the compression mechanism switches to the superior encoding despite the penalty of $c(E_1, E_2)$ in anticipation of further growing x . As $|x|$ goes to infinity $c(E_1, E_2)$ will be marginalized. Such a general compression engine \hat{E} cannot exist: It would have to compute $|E(x)|$ for *every possible* encoding E , in particular, \hat{E} is not simple. Instead it is computing the Kolmogorov complexity of x , which is known to be impossible.

As discussed in section 2.4.3, A 's size does not correspond to its Kolmogorov complexity. Therefore, sticking to simple, and hence computable, encodings we necessarily introduce a bias: Minimizing $|E_1(x)|$ needs not at all asymptotically coincide with minimizing the code length of $|E_2(x)|$, meaning that the learners will output different hypotheses depending on the encoding. Therefore MDL's objectivity is lost as soon as one introduces a fixed, objective encoding. In slightly different words, any encoding which enables direct implementation of MDL will necessarily come with a bias.

To put MDL in relation to the above definitions of learnability, it is safe to say that its limit criterion puts it in the neighborhood of IITL (see Section 3.2.1): Learning success may oscillate until the global optimum is found eventually, and it is unknown how long that will take. Said oscillation immediately rules out compatibility with the PAC setting (from Section 3.2.3). For MDL using full Kolmogorov complexity, statements about runtime or the amount of training data required clearly cannot be made. If one fixes a coding schema, this changes: One particular case will be treated in Section 4.2.

Proposition 3.14 claims that iterative compression of the data leads to the optimal solution wrt. to the bias introduced by the encoding. This does however not imply that a local, greedy search will suffice: Performance and simplicity of a hypothesis are not equivalent for the evaluation-based order $<_{D_T}$. MDL is only a *global* criterion meaning that while \hat{H} can be deemed optimal [VL00], simply finding shorter code lengths is not guaranteed to always improve performance [AV09]. This implies that approximations of Kolmogorov complexity cannot be used as a reliable guidance through the space of hypotheses [AJ06]. Indeed, as of now there is no known strategy which is guaranteed to solve the induction task without performing an exhaustive search.

The claim that a fixed encoding is strictly required becomes incorrect in the limit, when runtime is not seen as an issue. Enumeration of hypotheses, and in fact even encodings, will lead to a strictly optimal solution [Lev73]. This has been dubbed *Levin Search*. As a downside, such algorithms never terminate, and it cannot be decided how long it will take until they switch to another currently-best hypothesis (if ever). Still, this approach is *universal* for all well-defined, computable problems [Hut02] and strictly time-optimal for this very large class of problems [Sch03].

Despite those problems, and although there is much debate on the subject (mainly concerning the precise definition of *simplicity* [vS10]), there is extensive empirical evidence that the application of Occam's Razor in Grammar Induction can lead to generalization. This however is not backed by any stringent theoretical results, and in fact it should be easy to derive *No Free Lunch* situations [WM97] for any given language class.

4 Incremental Compression Algorithms

The idea of selecting a hypothesis consistent with the training data and incrementally compressing it is not new. Especially in the field of FSA induction this has become a widely used tool, mostly due to its intriguing theoretical features. This chapter will show how the design principles of FSA induction algorithms can be transferred to CFG learning. To this end, section 4.1 will review and explicate the assumptions and strategies of common FSA algorithms. Section 4.2 will take this as a starting point for the design of an MDL induction algorithm tailored towards CFGs.

4.1 State Merging Algorithms for FSAs

The learnability of regular languages has been studied extensively in the past, resulting in increasingly sophisticated implementations of the basic state merging algorithm [TB73]. The following section summarizes techniques widely used for finding a minimal FSA. Section 4.1.1 introduces choices for the learner's initialization. Section 4.1.2 motivates these, and presents theoretical results concerning the merging of states until a minimal FSA is found; section 4.1.3 considers the order in which current incremental state merging algorithms operate.

4.1.1 Initial Hypotheses

Within the domain of FSAs, *state merging algorithms* [TB73, OG92, Lan92] were designed as an answer to the following question: Assume a learner is presented with training data $D = \langle I_+, I_- \rangle$ consisting of two finite sets. I_+ contains sentences over an alphabet Σ that are accepted by a certain FSA A whereas I_- contains sentences that are not. How can the learner efficiently determine A or an equivalent FSA from D ?

Lacking further information, all the learner can do is to find an automaton which is consistent with D . In the spirit of Occam's razor (see section 3.3.1), if such an automaton is in some sense *minimal* it stands to reason that it is in fact very similar to A . As the name suggests, state merging algorithms focus on minimizing the number of a candidate FSA's states.

As mentioned in section 2.2 there exist algorithms which directly minimize a given FSA by computing its canonical counterpart [BBCF10]. The language accepted by this canonical FSA is still I_+ . Algorithms that do not generalize are of little help to the learning task: If the underlying language is infinite, an automaton accepting only a finite set of sentences cannot be correct.

The initial hypothesis of any state merging algorithm represents D , meaning it accepts precisely I_+ . There are two possible choices that are easily computed.

Definition 4.1. Given a language L , the *maximal canonical automaton* $MCA(L)$ is the smallest FSA with precisely one outgoing branch per $t \in L$, leading through t . All and only leaf nodes are accepting states.

Definition 4.2. Given a language L , the *prefix tree acceptor* $PTA(L)$ is the DFA where each state represents one unique prefix observed in L . All and only leaf nodes are accepting states.

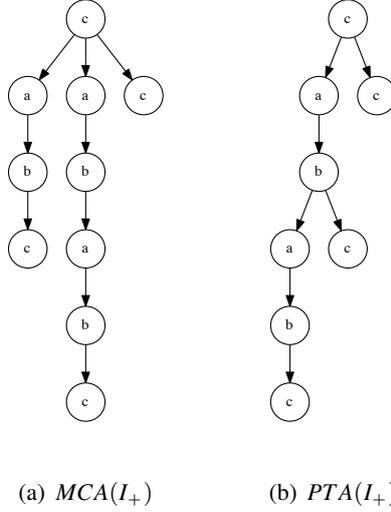


Figure 1: MCA and PTA of the set $I_+ = \{cab, cababc, cc\}$.

The intuitively simplest automaton to represent I_+ is $MCA(I_+)$. Its starting state has one outgoing branch per $t \in I_+$, and is therefore usually nondeterministic. $PTA(I_+)$ is the deterministic (but not canonical) version of $MCA(I_+)$. Figure 4.1.1 illustrates the two. Up to this stage no inference has happened. The issue of the initial hypothesis will be revisited shortly, in the meantime it is not important which one the learner employs. In the following the learner will work on the hypothesis FSA $H = \langle \Sigma, S, s, \delta, F \rangle$.

4.1.2 The Search Space of FSA Induction

Definition 4.3. Given a language L and an FSA $\langle \Sigma, S, s, \delta, F \rangle$, two states $s_1, s_2 \in S$ are *compatible* if for all $t \in I_+ \cup I_-$, for all $i \leq |t|$ we find $\delta^*(s_1, t[i :]) \in F$ if and only if $\delta^*(s_2, t[i :]) \in F$.

That is, states are compatible if every suffix in the training data leads to the same result, acceptance or rejection. State merging algorithms incrementally reduce the number of states by repeatedly merging two compatible states in S . The *merge* operation itself enhances s_1 with all incoming and outgoing edges of s_2 . Since it has become redundant, s_2 is subsequently discarded. More precisely, the merge results in the automaton $merge(H, s_1, s_2)$ as computed by algorithm 1.

Importantly, such merges are consistent with the training set and at the same time reduce the FSA's size. Counting shows that $M(H) \leq M(merge(H, s_1, s_2))$ with strict inequality whenever $s_1 \neq s_2$. The merge operation will not maintain a tree structure if H has one, and will often introduce nondeterminism in the FSA. This is not problematic as described in section 2.2.

Performing such merge operations in order to find a minimal DFA is only justifiable if one can show that the merges do not contradict available data and that they in fact can lead to the un-

Algorithm 1 The algorithm to compute $merge(\langle \Sigma, S, s, \delta, F \rangle, s_1, s_2)$.

Require: $Compatible(s_1, s_2)$

```

 $S' \leftarrow S \setminus \{s_2\}$ 
 $\delta' \leftarrow \delta \cup \{\langle s_1, \sigma, s' \rangle \mid \langle s_2, \sigma, s' \rangle \in \delta\}$ 
 $\delta' \leftarrow \delta' \cup \{\langle s', \sigma, s_1 \rangle \mid \langle s', \sigma, s_2 \rangle \in \delta\}$ 
if  $s = s_2$  then
  return  $\langle \Sigma, S', s_1, \delta', F \rangle$ 
else
  return  $\langle \Sigma, S', s, \delta', F \rangle$ 
end if

```

derlying FSA A . The first item is ensured by requiring that s_1, s_2 be compatible before a merge. Establishing the second item is more involved. While we previously argued with Occam's razor, it can be formally shown that for certain D , the merge procedure is a *correct* method of finding the target automaton.

Definition 4.4. Given two equivalent automata $A = \langle \Sigma, S, s_1, \delta, F \rangle$ and $A' = \langle \Sigma, S', s'_1, \delta', F' \rangle$, A is at least as *fine* as A' ($A \geq A'$) if for every $s' \in S'$ there is an $s \subseteq S$ such that

- $\{\langle \sigma, s'' \rangle \mid \{\langle s', \sigma, s'' \rangle \in \delta\} = \bigcup_{s \in s} \{\langle \sigma, s'' \rangle \mid \{\langle s, \sigma, s'' \rangle \in \delta\},$
- $\{\langle s', \sigma \rangle \mid \{\langle s'', \sigma, s' \rangle \in \delta\} = \bigcup_{s \in s} \{\langle s'', \sigma \rangle \mid \{\langle s'', \sigma, s \rangle \in \delta\},$ and
- $s' \in s.$

In other words, H is finer than H' if H' is the result of one or more merge operations performed on H ; in particular, the merge operation reduces fineness and hence size. It is clear that \geq is transitive, reflexive and antisymmetric. This imposes a partial order on the set $\mathcal{M}(H)$ of automata that can be constructed of H by merging. In particular, $Lat(MCA(I_+)) = \langle \mathcal{M}(MCA(I_+)), \geq \rangle$ is a lattice where the top element is the initial hypothesis $MCA(I_+)$ and the bottom element is the universal automaton.⁷ It follows that the merge process is only helpful to the learning process if A is a member of this lattice. It has been shown that under certain assumptions about the training data this is indeed the case.

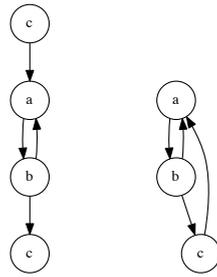
Definition 4.5 ([DMV94]). Given an FSA $A = \langle \Sigma, S, s, \delta, F \rangle$, a sample $I_+ \subseteq L(A)$ is *structurally complete* if during the testing whether $t_1, \dots, t_{|I_+|} \in L(A)$ every transition in δ can be exercised and every $f \in F$ can be reached at least once.

Theorem 4.6 ([DMV94]). *If I_+ is structurally complete wrt. A then $A \in Lat(MCA(I_+))$.*

Theorem 4.7 ([DMV94]). *If I_+ is structurally complete wrt. the canonical FSA it was sampled from, then this canonical FSA is in $Lat(PTA(I_+))$.*

Since $MCA(I_+) \geq PTA(I_+)$, it follows that $Lat(PTA(I_+)) \subseteq Lat(MCA(I_+))$. However, theorem 4.6 cannot be strengthened to the PTA, which would be desirable from a computational

⁷This is assuming that for every $\sigma \in \Sigma$ there is a transition from s_1 .



(a) The correct FSA. (b) Merging the two c-nodes leads to a different result.

Figure 2: Unfortunate merging within $MCA(\{cabac, cababc, cc\})$ can lead to degenerate FSAs.

complexity perspective. A first, naive attempt to infer the automaton from training data $\langle I_+, I_- \rangle$ is to construct the lattice of merges resulting from the initial hypothesis, while constraining the merge operation to compatible nodes. The hypothesis (hypotheses) with the greatest distance from the initial hypothesis in this lattice is the best guess for the underlying A . We call this approach *naive* because it is computationally not feasible. The following section presents methods to lower the computational requirements.

4.1.3 Runtime and Heuristics

When the inference algorithm is initialized with either the MCA or PTA of I_+ the number of possible node merges is quadratic in the total length of sentences in I_+ . As merging each pair would gain the same decrease in complexity (*i.e.*, rid the hypothesis H of one state) no preference can be given. This many possibilities render a complete search of $Lat(H)$ intractable.

Due to the incremental nature of state merging algorithms unfortunate merges carry over to future decisions, snowballing to potentially great deviations from the target DFA. Figure 4.1.3 continues the example from above. Each merge introduces new constraints on future merges, and one single wrong constraint suffices to have the learner never reach the optimal automaton. This becomes a bigger problem with increasing data sparseness.

At the same time computational requirements quickly increase with the number and length of strings in I_+ , making computationally easy strategies an even more pressing issue. It is therefore important to determine a good order of the merges. Historically, breadth-first search provided the most obvious solution [TB73, OG92] to the diverse challenges of DFA induction [Hin01]. Going one step further, the *Blue-Fringe* algorithm [dlHOV96, LPP98] provides a local breadth-first search mechanism. The key property is that it guarantees that one of the nodes merged is always the root of the tree. This results in a particularly fast algorithm which achieves identification of

A in polynomial time when provided with data [dlHOV96].

Limiting the Search Space Blue-Fringe as used in [JP98] works by partitioning the nodes of H into three distinct sets: red, blue and white nodes. Red nodes are mutually incompatible and cannot be merged. Blue nodes are reachable in one step from one of the red nodes and currently candidates for the merge procedure. White nodes are not yet inspected. For example, in case of the PTA, blue nodes are the roots of otherwise white labeled trees.

The two actions performed by Blue-Fringe are:

- Promote a blue node to red if it is unmergeable with any red node, and
- Merge a blue node with a red node.

The start state is always labeled red. During runtime, Blue-Fringe first produces all automata from merging compatible blue and red nodes. These are the candidate hypotheses for the next iteration. Next, for each candidate hypothesis, all blue nodes that cannot be merged with any red node are labeled red. All of its successors are labeled blue. This process is repeated until only red nodes are left, and after exhausting the search space, the candidate hypothesis (hypotheses) with the greatest least number of nodes is (are) returned.

Even when using Blue-Fringe, in many cases there is more than one option for a next merge, and exploring all possible sublattices is still very costly. Ideally, one should be able to compare and evaluate the likely gain of all these options without fully exploring them. This is done by heuristics.

Heuristics on Merge Candidates Blue-Fringe imposes a fixed order on the merges. While this order seems intuitively plausible, it is clearly not the single best strategy for all possible learning problems. Instead, performing merges first which are suggested by *heuristics* is a more flexible approach. The mainstream method is to use *evidence-driven state merging* (EDSM) [dlHOV96]. It inspects the training data and performs merges exactly in order of the amount of evidence, rather than in a predetermined order that hopefully correlates with that quantity. Evidence in Grammar Induction is the number of times a state is passed during testing the sentences in I_+ , so EDSM merges two oft-used states before less often visited states. By applying heuristics, the breath-first search is replaced by a best-first approach.

Due to data sparseness it is unlikely that there is sufficient information for EDSM to work unconditionally. Another possible heuristic is MDL: The merge which minimizes the description length is to be preferred. This idea however can only work if the $K(D | H)$ is taken into account, in which case it is proportional to EDSM. If this data-to-model code is not taken into account, every merge will reduce the description length of the hypothesis by an equal amount. This is important to keep in mind as we progress to CFG induction.

EDSM has a statistical flavor to it, and there is a large body of work using statistical methods to infer DFAs. As exemplary publication, and as starting points for discovering this wide field, [SJ03] deals with state merging for noisy data. More related to the information-theoretic MDL approach is [Tho00], which attempts to minimize the aforementioned Kullback-Leibler

divergence.

4.2 Iterative Compression Algorithms for CFGs

Facing the success of state merging methods for FSAs, the question if they can be transferred to stronger automata classes arises naturally. This section describes how the ideas underlying FSA induction can be adjusted and extended to handle CFGs. In particular, section 4.2.1 revisits the issue of measuring a grammar's size before section 4.2.2 presents feasible initial hypotheses for CFG learners. Section 4.2.3 will present operations on these hypotheses and argue that incremental CFG induction is a search through a lattice spanned by an initial hypothesis and those operations. An induction algorithm for CFGs will be derived, and its relevant properties will be presented. Section 4.2.4 will describe the runtime, and revisit the idea of heuristics.

A very early precursor of the ideas presented henceforth is [VB87], describing the version space of CFGs for a given language. Also, some of the results presented here have been implicitly assumed in the few works focussing on CFG induction [LS00, PPK⁺04], but never been spelled out to the best of my knowledge.

4.2.1 Size Measures Revisited

As before a learner is presented with training data $D = \langle I_+, I_- \rangle$ consisting of two finite sets of sentences that are generated by a CFG A and that are not, respectively. The learner must effectively determine A or an equivalent just from D .

The rationale behind state merging algorithms for FSAs remains unchanged in this new context: Assuming that D contains enough information, the smallest grammar consistent with the training data is indeed expected to be the underlying grammar, and as a result generalizes well. Thus, the CFG induction can be achieved by an algorithm which starts from some hypothesis H and proceeds to minimize the size of H as far as possible. While the size of FSAs can be approximated by their number of states (see section 4.1.2), we will first show that defining the size of a CFG as the number of its rewrite rules cannot work.

Theorem 4.8. *Given a positive sample I_+ , constructing a CFG $G = \langle N, \Sigma, R, s \rangle$ such that $L(G) = I_+$ which minimizes $|R|$ does not solve the induction task.*

Proof. For a sentence t , the smallest grammar G such that $L(G) = \{t\}$ requires only the rule $S \Rightarrow t$. For a set I_+ of sentences, construct one such rule per sentence: No CFG with fewer states will be equivalent to this, yet no generalization has happened. \square

This result is due to CFGs' capability to generate whole sequences by applying just one rule. Instead, we will have to take the more complex measure from definition 2.7.

Definition 4.9. Given a CFG $G = \langle N, \Sigma, R, s \rangle$, a rule $r \in R$ of the form $n \Rightarrow \beta$ is *redundant* if

1. r is never needed in parsing any of $L(G)$, or

2. $|\beta| = 1$ and $\beta \in N$ (so-called unit productions), or
3. there is another $r' \in R$ of the form $n' \Rightarrow \beta$, or
4. there is no other $r' \in R$ of the form $n \Rightarrow \beta'$ and $n \neq s$.

We call a rule of the form $n \Rightarrow n'$ (as in the second case) a *chain rule*.

Definition 4.10. Given a CFG $\langle N, \Sigma, R, s \rangle$, two rules $r_1, r_2 \in R$ of the form $n_1 \Rightarrow \beta$ and $n_2 \Rightarrow \gamma$ contain *redundancy* if there is a sequence u with $|u| \geq 2$ such that $u \sqsubseteq \beta$ and $u \sqsubseteq \gamma$.

Lemma 4.11. Given a CFG $G = \langle N, \Sigma, R, s \rangle$, adding redundant rules to R increases $M(G)$.

Proof. We will show that redundancy strictly increases its size, following the order of definition 4.9.

1. Add two new symbols n_1, n_2 to N and a rule $n_1 \Rightarrow n_2$ to R . This increases $M(G)$ by 2.
2. Take some $n_1, n_2 \in N$ and add a new symbol n' to N , then add the rules $n_1 \Rightarrow n'$ and $n' \Rightarrow n_2$ to R . This increases $M(G)$ by 4.
3. Take a rule $n \Rightarrow \beta$ from R and add a new symbol n' to N . Form the rule $n' \Rightarrow \beta$ and replace some occurrences of n within R by n' . This increases $M(G)$ by $|\beta| + 1$.
4. Take a rule $r \equiv n \Rightarrow \beta\gamma$ from R . Adding a new rule $n' \Rightarrow \beta$ to R using a new symbol n' , and changing r to $n \Rightarrow n'\gamma$ increases $M(G)$ by 2. \square

Lemma 4.12. Given a CFG $G = \langle N, \Sigma, R, s \rangle$, adding redundancy to $r_1, r_2 \in R$ generally increases $M(G)$.

Proof. Wlog., let R contain n rules $r_i \equiv n_i \Rightarrow n'\gamma_i$ and let $r' \equiv n' \Rightarrow \beta \in R$. Hard coding r' into the r_i increases $M(G)$ by $n \cdot (|\beta| - 1) - (|\beta| + 1)$, hence $M(G)$ grows as soon as $(n - 1) \cdot |\beta| - n - 1 > 0$. \square

These results show that the definition of redundancy in the rule set is equivalent to canonicity (definition 2.8):

Theorem 4.13. A CFG is canonical if it contains no redundant rules, and no rules that contain redundancy.

Proof. Combining lemmata 4.11 and 4.12, it is easy to show by induction on the structure of a rule that removing redundant rules in R , or just removing redundancy within rules, generally reduces the size of the grammar. In the cases of definition 4.10 where $M(G)$ does not shrink, it does not increase either. Hence the canonical grammar is one of the grammars which achieve minimal size. \square

4.2.2 Initial Hypotheses

There are two approaches to CFG induction that intuitively look promising. Either one starts with a very flat grammar and constructs new and smaller rewrite rules out of the necessarily very big ones, or one starts with many rules and exploits structural similarities between trees they span. In other words, one either wants to produce a grammar with as few or as many rules as possible while avoiding redundancy. In the following this will be referred to as the bottom-up and top-down approach, respectively.

The initial hypothesis H will ideally be easily computed. In any case it must be consistent with the training data, meaning $L(H) = I_+$ and $L(H) \cap I_- = \emptyset$. The following two definitions capture the initial hypotheses for the bottom-up and top-down approaches.

Definition 4.14. The *sentence acceptor* $SA(L)$ of a language L is $\langle \{S\}, L, \{S \Rightarrow t \mid t \in I_+\}, S \rangle$.

Definition 4.15. The *maximum canonical grammar* $MCG(L)$ of a language L is $\langle N, \{t_i \in t \mid t \in L\}, R, S \rangle$ where N and R are such that $MCG(L)$ models exactly all n -ary parse trees over L without containing chain rules, for $n \in \mathbb{N}$.

It is easy to verify that $SA(I_+)$ will generally contain rules hosting redundancy, but no completely redundant rules. $MCG(I_+)$ will have both.

In the interest of easy computability it is worth investigating the computational requirements for the SA and MCG. Note that $SA(L)$ has already been used in the proof of Theorem 4.8; it is obvious that it can be computed in $O(|I_+|)$ time.

Theorem 4.16. *For a sentence of length n there exist exponentially many b -ary parse trees.*

Proof. We first focus on the simple case where the size b of rules is fixed.⁸ Let n be the length of the sentence in question, so the height of the parse tree is $\lceil \log_b(n) \rceil$.

During incremental bottom-up construction of a parse tree, each round will see b adjacent items grouped to a nonterminal. There are $n - (b - 1)$ possibilities to do so. The resulting sequence has $n - b + 1$ items; the process terminates once the sequence has length 1. As a result, there are a total of

$$\sum_{i=1}^{\lceil \log_b(n) \rceil} n - i \cdot (b - 1)$$

possibilities. Hence for arbitrary but fixed values of b this number increases to

$$\prod_{b=2}^n \sum_{i=1}^{\lceil \log_b(n) \rceil} n - i \cdot (b - 1),$$

which is $O(2^n)$. □

Corollary 4.17. *$MCG(L)$ cannot be computed in polynomial time.*

⁸Except in the case of the rule expanding the start symbol S , which may be smaller.

Since I_+ will usually be large, this immediately disqualifies its MCG as a possible initial hypothesis, suggesting that the sentence acceptor must be used instead. In the following an algorithm will be presented which by a series of transformations constructs the canonical equivalent of $SA(I_+)$.

4.2.3 The Search Space of CFG Induction

The Grammar Induction algorithm discussed henceforth will take a CFG H and iteratively expose redundancies within the grammar. It is only concerned with the type of redundancy described in definition 4.10. The central method *RuleSplit* described in algorithm 2 removes one redundant substring from the entire rule set, replacing it by a novel nonterminal symbol \aleph . It further extends R by a rule expanding \aleph to the newly removed substring. This does not change the language generated by H and, by lemma 4.12, generally decreases the size of H .

Leaving negative training data aside for now, *RuleSplit* makes use of the function *substitute* which takes a string $u \sqsubseteq (N \cup \Sigma)^*$, a nonterminal symbol n and a rewrite rule r as input and transforms the rewrite rule: Every occurrence of u in r is replaced by n in a left-to-right manner.

Algorithm 2 *RuleSplit*(N, R, u)

```

 $N \leftarrow N \cup \{\aleph\}$  //  $\aleph$  is a new symbol  $\notin N$ 
 $R \leftarrow R \cup \{\aleph \Rightarrow u\}$ 
for all  $n \Rightarrow \beta \in R$  such that  $u \sqsubseteq \beta$  do
     $R \leftarrow R \cup \{\text{substitute}(u, \aleph, n \Rightarrow \beta)\}$ 
end for
 $R \leftarrow R \setminus \{n \Rightarrow \beta\}$ 
return  $\langle N, R \rangle$ 

```

Example 4.18. Take a toy grammar $\langle \{S\}, \{0, 1\}, \{S \Rightarrow 111000, S \Rightarrow 100100\}, S \rangle$. This grammar contains redundancies, for example $u \equiv 100$: We have $u \sqsubseteq 111000$ and $u \sqsubseteq 100100$. Using *RuleSplit*, this redundancy can be eliminated: Computing $\text{RuleSplit}(\{S\}, \{S \Rightarrow 111000, S \Rightarrow 100100\}, u) = \langle \{S, \aleph\}, \{S \Rightarrow 11\aleph 0, S \Rightarrow \aleph\aleph, \aleph \Rightarrow 100\} \rangle$ finds the smaller grammar $\langle \{S, \aleph\}, \{0, 1\}, \{S \Rightarrow 11\aleph 0, S \Rightarrow \aleph\aleph, \aleph \Rightarrow 100\}, S \rangle$.

In order to detect of as much redundancy as possible, which will eventually lead to the greatest generalization, the hardly subtle function *SplitAll* described in algorithm 3 performs an exhaustive search over all substrings found in right hand sides of rules in R ; if redundancy is detected it is removed by means of *RuleSplit*. At this stage the redundancy in the grammar decreases again because only one single nonterminal is introduced in total (as opposed to using new $\aleph_1, \aleph_2, \dots$ each round). In order to not lose information during the extraction process, substrings are searched for in the positive training data I_+ rather than the rule set of the grammar. The transformation starts with the longest substrings first, and is performed only if the resulting grammar has in fact less redundancy. This algorithm makes use of the function $\#(t, X)$ which counts the elements in X which t is a substring of without overlap.

Example 4.19. Let $I_+ = \{111000, 100100\}$, so constructing the sentence acceptor yields the grammar from example 4.18. *SplitAll* starts out by finding all substrings shared between sentence in I_+ , which are 100, 10, and 00. Starting with the longest one, the redundancy is removed in *RuleSplit*-fashion, yielding $\langle \{S, \aleph\}, \{0, 1\}, \{S \Rightarrow 11\aleph 0, S \Rightarrow \aleph\aleph, \aleph \Rightarrow 100\}, S \rangle$ as current grammar. The algorithm then moves on to the next shorter substring, *i.e.*, 10. Clearly there are rules whose body share this substring ($S \Rightarrow 111000$ and $\aleph \Rightarrow 100$). However, 100 has been worked on before, and it occurred in I_+ three times. As 01 does not occur more often, it is ignored. The same holds for the string 00.

Algorithm 3 *SplitAll*(I_+)

```

 $\langle N, \Sigma, R, S \rangle \leftarrow SA(I_+)$ 
 $seen \leftarrow \emptyset, out \leftarrow \emptyset$ 
 $U \leftarrow \{u \mid \exists t_1, t_2 \in I_+ : t_1 \neq t_2 \wedge u \sqsubseteq t_1 \wedge u \sqsubseteq t_2 \wedge |u| \geq 2\}$ 
 $i \leftarrow \max_{t \in I_+} |t|$ 
 $N \leftarrow N \cup \{\aleph\}$ 
while  $i \geq 2$  do
  for all  $u \in U$  such that  $|u| = i$  do
    if not  $\exists \langle u', x \rangle \in seen$  such that  $u \sqsubseteq u' \wedge x = \#(u', I_+)$  then
       $seen \leftarrow seen \cup \langle u, \#(u, I_+) \rangle$ 
       $R \leftarrow R \cup \{\aleph \Rightarrow u\}$ 
      for all  $n \Rightarrow \beta \in R$  such that  $u \sqsubseteq \beta$  do
         $R \leftarrow R \cup \{substitute(u, \aleph, n \Rightarrow \beta)\}$ 
         $out \leftarrow out \cup \{n \Rightarrow \beta\}$ 
      end for
    end if
  end for
   $i \leftarrow i - 1$ 
end while
 $R \leftarrow R \setminus out$ 
return  $\langle N, \Sigma, R, S \rangle$ 

```

The *RuleSplit* method is very similar to the Sequitur algorithm [NmW97]. However in extracting all rules, as opposed to one linear order of extraction, *SplitAll* is much more powerful, at the cost of the linear runtime of Sequitur.

At this point, a massive generalization has taken place. As every new rule re-uses the same head, *SplitAll*(A) will generate many more sentences than A . This overgeneralization is due to the negligence of the negative training data I_- . To only generate grammars that are functionally equivalent to $D = \langle I_+, I_- \rangle$ the *SplitAll* algorithm must be enhanced.

Definition 4.20. Given a language L and a CFG $\langle N, \Sigma, R, S \rangle$, define $R' = R \cup \{n_1 \Rightarrow \beta \mid n_2 \Rightarrow \beta \in R\} \cup \{n_2 \Rightarrow \beta \mid n_1 \Rightarrow \beta \in R\}$ for $r_1, r_2 \in R$. Then r_1, r_2 are *compatible* if $I_+ \sqsubseteq L(\langle N, \Sigma, R', S \rangle)$ and $L(\langle N, \Sigma, R', S \rangle) \cap I_- = \emptyset$.

That is, we call two rules compatible if they can be merged by interchanging their heads without

losing functional equivalence of the CFG to D . The following algorithm 4, *LearnCFG*, adheres to negative training data by introducing new terminal symbols whenever needed. It does so by reusing as many terminal symbols as possible, only creating a new one if there are none to use as head without giving up functional equivalence to D .

The main change from *SplitAll* is that before reusing any nonterminal symbol, it is ensured that no negative training samples can be produced by this new rule. At the same time, as many nonterminals are reused as possible to guarantee maximal generalization. Only if no compatible nonterminals are present, an entirely new one is created.

Algorithm 4 *LearnCFG*($\langle I_+, I_- \rangle$)

```

 $\langle N, \Sigma, R, S \rangle \leftarrow SA(I_+)$ 
 $seen \leftarrow \emptyset, new = \emptyset, out \leftarrow \emptyset$ 
 $U \leftarrow \{u \mid \exists t_1, t_2 \in I_+ : t_1 \neq t_2 \wedge u \sqsubseteq t_1 \wedge u \sqsubseteq t_2 \wedge |u| \geq 2\}$ 
 $i \leftarrow \max_{t \in I_+} |t|$ 
 $N \leftarrow N \cup \{\mathfrak{N}\}$ 
while  $i \geq 2$  do
  for all  $u \in U$  such that  $|u| = i$  do
    if not  $\exists \langle u', x \rangle \in seen$  such that  $u \sqsubseteq u' \wedge x = \#(u', I_+)$  then
      if  $\exists \mathfrak{N} \in new : \forall \mathfrak{N} \Rightarrow \beta \in R : compatible(\mathfrak{N} \Rightarrow \beta, \mathfrak{N} \Rightarrow u, \langle I_+, I_- \rangle)$  then
        for all  $\mathfrak{N} \in new : \forall r \in \{\mathfrak{N} \Rightarrow \beta \in R\} : compatible(r, \mathfrak{N} \Rightarrow u, \langle I_+, I_- \rangle)$  do
           $seen \leftarrow seen \cup \langle u, \#(u, I_+) \rangle$ 
           $R \leftarrow R \cup \{\mathfrak{N} \Rightarrow u\}$ 
          for all  $n \Rightarrow \beta \in R$  such that  $u \sqsubseteq \beta$  do
             $R \leftarrow R \cup \{substitute(u, \mathfrak{N}, n \Rightarrow \beta)\}$ 
             $out \leftarrow out \cup \{n \Rightarrow \beta\}$ 
          end for
        end for
      end if
    else
       $seen \leftarrow seen \cup \langle u, \#(u, I_+) \rangle$ 
       $N \leftarrow N \cup \{\mathfrak{N}'\}$  //  $\mathfrak{N}'$  is a new symbol  $\notin N$ 
       $R \leftarrow R \cup \{\mathfrak{N}' \Rightarrow u\}$ 
      for all  $n \Rightarrow \beta \in R$  such that  $u \sqsubseteq \beta$  do
         $R \leftarrow R \cup \{substitute(u, \mathfrak{N}', n \Rightarrow \beta)\}$ 
         $out \leftarrow out \cup \{n \Rightarrow \beta\}$ 
      end for
    end if
  end if
  end for
   $i \leftarrow i - 1$ 
end while
 $R \leftarrow R \setminus out$ 
return  $\langle N, \Sigma, R, S \rangle$ 

```

Lemma 4.21. $I_+ \subseteq L(\text{LearnCFG}(\langle I_+, I_- \rangle))$.

Proof. The claim is obvious for $\text{SplitAll}(I_+)$. As LearnCFG is really just a modified version of SplitAll , it suffices to inspect the modification. Observe that even if no compatible \mathfrak{N}_i can be found, the rule to be added is not discarded but merely uses a new nonterminal symbol. Hence the claim follows. \square

This establishes consistency with positive training data, but it does not explain to what extent the actual rules of the target grammar can be learned. To investigate this, we observe that the RuleSplit operation, applied iteratively to an initial hypothesis, spans a lattice of CFGs. This lattice contains CFGs consistent with I_+ . The bottom element is the sentence acceptor of I_+ , the top element is a universal grammar over the target grammar's alphabet. In other words, higher elements of the $\text{Lat}(\text{SA}(I_+))$ are increasingly deprived of redundancy.

Definition 4.22. Given a CFG $A = \langle N, \Sigma, R, S \rangle$, a sample $I_+ \subseteq L(A)$ is *structurally complete* if during testing whether $t_1, \dots, t_{|I_+|} \in L(A)$ every rewrite rule in R must be exercised twice.

A rewrite rule r will be created by RuleSplit if the affected substring occurs at least twice in I_+ . This is precisely the condition for structural completeness. On the other hand, if r is only exercised once, the substring it creates has no matching counterpart in I_+ and r will pass undetected by RuleSplit .

Theorem 4.23. *If and only if I_+ is structurally complete wrt. the CFG A it was sampled from then $A \in \text{Lat}(\text{SA}(I_+))$.*

The natural follow-up question is concerned with the influence of negative training data. For a given target grammar A , is there a set I_- which stops RuleSplit from exploring grammars which generalize too much? The solution looks as follows.

Definition 4.24. Given a canonical CFG $A = \langle N, \Sigma, R, S \rangle$, a sample $I_- \subseteq \overline{L(A)}$ is *structurally complete* if for every $n \Rightarrow \beta$ seen in $\text{SplitAll}(I_+)$, there is a sentence in I_- that is only parseable using $n \Rightarrow \beta$ whenever $L(\langle N, \Sigma, R \cup \{n \Rightarrow R\}, S \rangle) \cap \overline{L(A)} \neq \emptyset$.

In words, the structurally complete negative sample set has a counterexample for every rule-body combination that can be produced using rules of A . The effect is immediately obvious: $\text{Lat}(\text{SA}(I_+))$ will have all elements removed that disagree with I_- . In particular, the canonical grammar will not be removed.

It is possible to test if a certain rule r is required to parse a sentence. First, we apply the CYK parsing algorithm with the original grammar. Afterwards we remove r from the grammar and attempt to parse the sentence again. What is problematic however is that we already may have introduced a set of rules which can function as a substitute of r . If $r \equiv n \Rightarrow \beta$, this amounts to having multiple rules r_1, r_2, \dots with bodies β_1, β_2, \dots such that $\beta \equiv \beta_1 \beta_2 \dots$. All of these rules were introduced during the learning phase, and justified by regularity in the positive training data. Hence there is a conflict between generalization and maintaining consistency with I_- , and it is not clear how to solve the conflict. The best we can claim about LearnCFG is hence:

Corollary 4.25. *Given structurally complete training data D sampled from a grammar A , $L(\text{LearnCFG}(D)) \supseteq L(A)$.*

We can hence not give a positive answer to the main question of Grammar Induction as posed in section 3.1. The algorithm *LearnCFG* is able to extract all generalization from the positive training data, but generalization is not sufficiently contained by negative training data. The next section is dedicated to an analysis of the runtime requirements of *LearnCFG*.

4.2.4 Runtime and Heuristics

To analyze the runtime of this CFG induction algorithm, we will examine the individual building blocks of *LearnCFG*. Firstly, the heart of the induction algorithm is *RuleSplit* (algorithm 2). Just like the *merge* operation in the case of FSAs (algorithm 1) it is easily computed in $O(|R| \cdot \max_{n \Rightarrow \beta \in R} |\beta|)$ time: The algorithm needs to iterate through all rules in the rule set R it is presented with. It follows that the exhaustive substitutions which *SplitAll* (algorithm 3) performs can be done in time polynomial in the size of the target grammar.

However, before substituting, all shared substrings in the positive training data must be found. This is a variant of the *longest common substring* problem [BHR00], for which a brute-force approach is computationally infeasible. Using dynamic programming techniques, the runtime can be made polynomial in the sentences' length [Wan07].

The final algorithm 4, *LearnCFG*, introduces a consistency check with I_- . This brings a quadratic increase of runtime. All in all, it can be said that *LearnCFG* runs in time polynomial to the size of its input $\langle I_+, I_- \rangle$. As it fails to reliably identify the target grammar, this is not in contradiction to the impossibility of polynomial time identification [Pit89] from section 3.2.2.

Still, this is a stark contrast to FSA learning algorithms, which only can achieve polynomial runtime if they do not explore their full search space $\text{Lat}(MCA(I_+))$. Naturally, visiting every grammar in $\text{Lat}(SA(I_+))$ also takes exponential time. The reason this is indeed unnecessary lies in the order in which *RuleSplit* is applied: By working on the longest common substrings first eliminates the need to inspect all substrings of already extracted subsequences. The ability of CFGs to generate full strings in one therefore brings increased complexity, but at the same time suggests a correct way of pruning of the search space. It therefore also makes heuristics unreasonable since obviously, every extraction of a new rule is necessary for learning the target grammar.

5 Properties and Applications of RuleSplit Grammars

5.1 Learning by Compression?

Section 3.3.3 gave a detailed description of the promises made by the theory of MDL. In particular, proposition 3.14 expressed the hope that an iterative compression will lead to the correct solution. In this respect it is worth evaluating whether *LearnCFG* is an, in this sense, proper MDL algorithm.

First and foremost, the algorithm has been designed to minimize the size measure M of context-free grammars as defined in definition 2.7. This measure should satisfy the requirements of being *reasonable* [Pit89]. While M certainly does not resemble Kolmogorov complexity, we showed in theorem 4.13 that minimizing M does lead to a canonical grammar. We argued that traversing a lattice of some initial hypothesis and a complexity-reducing operation should lead to the correct solution. The *sentence acceptor* from definition 4.14 is a straightforward way of computing a grammar consistent with the training data $\langle I_+, I_- \rangle$. And from the algorithm itself it is clear that generalization will take place whenever there is evidence in I_+ and I_- does not block it. So *LearnCFG* attempts to implement MDL with a *simple* encoding. As discussed in length, there can be no algorithm which actually implements the ideal form of MDL due to the incomputability of Kolmogorov complexity.

The following shows that *LearnCFG*, however, does not compress linearly. From lemma 4.12 it is known that the *RuleSplit* procedure reduces redundancy in a grammar. A fortiori:

Lemma 5.1. *If a grammar A contains no redundant rules then $SplitAll(A)$ will not contain redundant rules either.*

Proof. For a redundant rule r , we follow the order of definition 4.9.

1. *SplitAll* never adds rules with right hand sides that *can* not be used for parsing I_+ . To see that it also never generates rules that *need* not be used, recall that whenever a new rule is created, the old substring is replaced by \mathfrak{N} , hence the new rule must be used to parse the sentence in question. As substrings are only extracted if they lead to generalization, no „duplicate” rules will be generated.
2. As substrings targeted by *RuleSplit* are of length ≥ 2 no r with $|r| < 2$ will exist.
3. When a substring u is extracted during creation of a rule $\mathfrak{N} \Rightarrow u$ then u is replaced everywhere, so no second rule with this body will be created.
4. For every rule that is newly introduced, *RuleSplit* uses the same head and any existing nonterminals expanding to u will be merged with \mathfrak{N} . □

What about the case where the structurally complete set of training data has not been generated from a canonical grammar? In such grammars it is possible that a substring *aaabbb* is extracted via *RuleSplit*, followed by the substrings *aab* and *abb*. To guarantee correctness of the algorithm, both new rules must be kept. This leads to a temporary increase of the grammar’s size. A similar phenomenon occurs if negative training data forced the introduction of a new nonterminal

symbol. As a result, *LearnCFG* does not compress monotonically, and the results of [AV09] carry over to our approach. While *LearnCFG* learns by compression, it hence fails to meet the expectations of proposition 3.14.

Further backtracking to sections 3.2.1 through 3.2.3, the *LearnCFG* algorithm does not identify context-free languages in the limit, as it will overgeneralize. It has a polynomial runtime based on the analysis in 4.2.4. Adding more data will allow the algorithm to detect more rules of the target grammar, however the nonmonotonic manner in which the algorithm compresses the training data do not allow for PAC-results. As adding positive training data exposes *LearnCFG* to the risk of overgeneralizing more, no results resembling PAC-learnability can be established.

We argued that the *RuleSplit* procedure spans a lattice whose top element is the universal grammar. If negative samples are absent, *SplitAll* works to reach this top element, leaving only inseparable substrings untouched. In essence, it produces a universal grammar whose alphabet consists of strings whose substrings never appear isolated in the positive samples. It is hence clear that also the rules of the learnt grammar are mainly shaped by the negative examples, and less by the positive data.

5.2 Applications of *LearnCFG*

As is the case with any machine learning algorithm, the obvious question is: What is its area of application? Context-free grammars are a classical domain of language modeling. However, language modeling of natural language usually works on positive data only. This is problematic for two reasons.

First, natural language does not necessarily have a context-free structure. Second, it always comes with a *truth bias* which causes perfectly grammatical strings like “the quick sleeper” to be underrepresented in corpora. More severely however is the lack of negative training data. Even if one would randomly generate phrases, many of them would probably still sound grammatical or even make sense to a natural speaker. Combined with the overwhelming amounts of negative training data required, natural language seems a poor target for *LearnCFG*. Likewise, no claims about first language acquisition [Cla04] should be derived from this thesis. Its applications are therefore presumably found in areas with artificial languages with context-free forms, such as XML.

Finding similarities of multiple languages by comparing their respective compressibility has been attempted a number of times: [CD10] tries to define a CFG as a *similarity measure*. It attempts to find canonical grammars (relative to a different size measure), and then compares the length of derivations of strings under these grammars. This could also be done with *LearnCFG*, with greater theoretical justifiability. It is however doubtful how far this method can be taken: [KY00] used a greater set of learning operators, some of which explicitly increase the grammar’s size. Then, redundancy in the derivations are measured. As discussed earlier, this can only work correctly if full Kolmogorov complexity is used. Nevertheless, the wellfoundedness of *LearnCFG* can contribute to a better understanding of when such comparisons work.

5.3 Comparison to Other Algorithms

There are other standard algorithms for CFG induction in the literature. And while the objective of this thesis is not to present a new algorithm but rather to understand the principles behind CFG induction, it is still worthwhile to briefly outline similarities.

EMILE The EMILE toolbox [AV02b] starts out by generating all possible pairs first-order rules from the sentences in the training data. It then merges substitutable rules just like *LearnCFG* did. The main difference is the initial hypothesis: By focussing on first-order rules, a shallow version of the maximum canonical grammar (definition 4.15) is used. The fact that this can suffice to learn a grammar is remarkable but not obvious, and indeed holds for *shallow* CFGs only [Adr99].

Alignment-Based Learning Alignment-based learning (ABL) [vZ00] finds shared substrings in sentences, but then focuses on their *differences*. If unique substrings are found surrounded by equal contexts, they are merged into one constituent. ABL then removes overlapping constituents via evidence-based heuristics. This obviously is a very different approach, leaning more towards language modeling. A comparison to EMILE can be found in [vZA01].

Concluding, it can certainly be said that the *RuleSplit* approach taken in this paper is not entirely new, but this was to be expected as there are only so many ways in which useful information can be extracted from a corpus. However, the design choices in this thesis and the consequences of the proposed strict bottom-up way have to the best of my knowledge not been worked out in the literature.

6 Conclusion

This thesis presented a state-merging approach to context-free grammar learning. We derived a compression procedure as well as an initial hypothesis from the principles of FSA induction, thereby also explicating the main requirements of an MDL algorithm for Grammar Induction. The resulting induction algorithm, *LearnCFG*, is complete for the class of context-free languages. We provided an analysis of structurally complete data sets and showed that *LearnCFG* has a runtime polynomial in the lengths of the sentences in the training data.

The algorithm failed to behave in the way we had hoped for in proposition 3.14 (section 3): It compresses nonmonotonically and is tailored to a simple encoding rather than working on Kolmogorov complexity. While the latter is unavoidable, the former was at least to be expected due to similar results in the case of state-merging algorithms for finite state automata. We also noted how certain regularities within the positive training data can, when exploited, lead to violations of the negative training data. As there is no justifiable *a priori* reason to exclude some expressions of regularities while making use of others, this issue must remain open for the moment.

The most obvious next step is an implementation and comparison to other algorithms [KY00, NI00, AV02b, vZ00, CD10]. However, the emphasis of this thesis is on the general, and hence theoretic, requirements of such learning algorithms. An empirical evaluation might gain new insights on the effect and frequency of the conflicts within the data as described above. But it would have to rely on some kind of training data which will be biased towards one domain. Likewise, the extension to probabilistic CFG learning seems unnecessary as we have shown that heuristics are not required for *LearnCFG*. It would however be interesting to see how well artificial languages such as XML could be learned in practice.

The most interesting questions not answered in this thesis is: Is there an encoding which leads to faster generalization? As we argued in length, the particular choice of an encoding determines the bias and hence the path which the induction algorithm takes through the hypothesis space.

Furthermore, the work in this thesis could be extended to context-sensitive languages. Further research in this field could rid existing methods of its probabilistic flavor, if it again can be shown that heuristics can be avoided: The strongest machine learning algorithms are those which have an optimal, tractable and correct way of selecting ever better hypotheses.

References

- [AASV09] L. Antunes, Matos. A., A. Souto, and P. M. B. Vitányi. Depth As Randomness Deficiency. *Theory of Computing Systems*, 45(4):724 – 739, July 2009.
- [Adr99] Pieter W. Adriaans. Learning shallow context-free languages under simple distributions, 1999.
- [AJ06] Pieter W. Adriaans and Cerial Jacobs. Using MDL for grammar induction. In *ICGI*, pages 293–306, 2006.
- [AM10] Pieter W. Adriaans and Wico Mulder. MDL in the limit. In *ICGI*, pages 258–261, 2010.
- [Ang80] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.
- [AV02a] Pieter Adriaans and Marco Vervoort. The emile 4.1 grammar induction toolbox. In Pieter Adriaans, Henning Fernau, and Menno van Zaanen, editors, *Grammatical Inference: Algorithms and Applications*, volume 2484 of *Lecture Notes in Computer Science*, pages 714–718. Springer Berlin / Heidelberg, 2002.
- [AV02b] Pieter W. Adriaans and Marco Vervoort. The emile 4.1 grammar induction toolbox. In *ICGI*, pages 293–295, 2002.
- [AV09] Pieter W. Adriaans and Paul M. B. Vitányi. Approximation of the two-part MDL code. *IEEE Trans. Inf. Theor.*, 55(1):444–457, 2009.
- [BBCF10] Jean Berstel, Luc Boasson, Olivier Carton, and Isabelle Fagnot. Minimization of automata. *CoRR*, abs/1010.5318, 2010.
- [Beh00] Ralf Behrens. A grammar based model for xml schema integration. In *In British National Conference on Databases (BNCOD)*, pages 172–190, 2000.
- [BHR00] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE’00)*, Washington, DC, USA, 2000. IEEE Computer Society.
- [Bri10] Encyclopædia Britannica. Ockham’s razor, June 2010.
- [CD10] Daniele Cerra and Mihai Datcu. A similarity measure using smallest context-free grammars. In *Proceedings of the 2010 Data Compression Conference, DCC 2010*, pages 346–355, Washington, DC, USA, 2010. IEEE Computer Society.
- [CG96] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318, Morristown, NJ, USA, 1996. Association for Computational Linguistics.
- [Cho56] N. Chomsky. Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3):113 –124, 1956.

- [CKP03] Orlando Cicchello, Stefan C. Kremer, and Fernando Pereira. Inducing grammars from sparse data sets: A survey of algorithms and results, 2003.
- [Cla04] Alexander Clark. Grammatical inference and the argument from the poverty of the stimulus, 2004.
- [Coc69] John Cocke. *Programming languages and their compilers: Preliminary notes*. Courant Institute of Mathematical Sciences, New York University, 1969.
- [Coo02] Barry S. Cooper. *Computability Theory*. CRC Press, Inc., Boca Raton, FL, USA, 2002.
- [CV07] Nancy Chang and Umesh Vazirani. Learning natural language: A review of formal and computational approaches, 2007.
- [DDG96] François Denis, Cyrille D’Halluin, and Rémi Gilleron. Pac learning with simple examples. In *STACS ’96: Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, pages 231–242, London, UK, 1996. Springer-Verlag.
- [dlH95] Colin de la Higuera. Characteristic sets for polynomial grammatical inference, 1995.
- [dlH05] Colin de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2005.
- [dlH09] Colin de la Higuera. Learning finite state machines. In *FSMNLP*, pages 1–10, 2009.
- [dlH10] Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York, NY, USA, 2010.
- [dlHO04] Colin de la Higuera and Jose Oncina. Learning context-free languages, 2004.
- [dlHOV96] C. de la Higuera, Jose Oncina, and E. Vidal. Identification of DFA: data-dependent versus data-independent algorithms, 1996.
- [DMV94] P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference? In Rafael Carrasco and Jose Oncina, editors, *Grammatical Inference and Applications*, volume 862 of *Lecture Notes in Computer Science*, pages 25–37. Springer Berlin / Heidelberg, 1994.
- [Doy02] Jon Doyle. What is church’s thesis? An outline. *Minds Mach.*, 12:519–520, November 2002.
- [DPS02] Michael Domaratzki, Giovanni Pighizzini, and Jeffrey Shallit. Simulating finite automata with context-free grammars. *Inf. Process. Lett.*, 84:339–344, December 2002.
- [ES03] D. M. Endres and J. E. Schindelin. A new metric for probability distributions. *IEEE Trans. Inf. Theory*, 49:1858–60, 2003.
- [FB86a] K S Fu and T L Booth. Grammatical inference: introduction and survey — part i. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8:343–359, May 1986.
- [FB86b] K S Fu and T L Booth. Grammatical inference: introduction and survey — part ii. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8:360–375, May 1986.

- [Fer01] Henning Fernau. Learning xml grammars. In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, volume 2123 of *Lecture Notes in Computer Science*, pages 73–87. Springer Berlin / Heidelberg, 2001. 10.1007/3-540-44596-X7.
- [For09] Lance Fortnow. The status of the p versus np problem. *Commun. ACM*, 52:78–86, September 2009.
- [GAJT07] Jianfeng Gao, Galen Andrew, Mark Johnson, and Kristina Toutanova. A comparative study of parameter estimation methods for statistical natural language processing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 824–831, Prague, Czech Republic, June 2007. Association for Computational Linguistics.
- [Gol67] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.
- [Gol78] E. M. Gold. Complexity of automaton identification from given data. *Inform. Control*, 37:302–320, 1978.
- [GPW82] Jonathan Goldstine, John K. Price, and Detlef Wotschke. A pushdown automaton or a context-free grammar: Which is more economical? *Theoretical Computer Science*, 18(1):33 – 40, 1982.
- [Grü05] Peter Grünwald. A tutorial introduction to the minimum description length principle. In *Advances in Minimum Description Length: Theory and Applications*. MIT Press, 2005.
- [Hea87] Tom Head. Formal language theory and dna: An analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49:737–759, 1987. 10.1007/BF02481771.
- [Hin01] Philip Hingston. Inference of regular languages using model simplicity. In *ACSC '01: Proceedings of the 24th Australasian conference on Computer science*, pages 69–76, Washington, DC, USA, 2001. IEEE Computer Society.
- [HMU07] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Pearson Addison-Wesley, Upper Saddle River, NJ, 3rd edition, 2007.
- [Huf07] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, January 2007.
- [Hut02] Marcus Hutter. The fastest and shortest algorithm for all well-defined problems. *International Journal of Foundations of Computer Science*, 13(3):431–443, June 2002.
- [Imm83] Neil Immerman. Languages which capture complexity classes. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 347–354, New York, NY, USA, 1983. ACM.
- [JM08] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice

Hall, 2 edition, 2008.

- [Joh98] Mark Johnson. The DOP estimation method is biased and inconsistent. *Computational Linguistics*, 28:71–76, 1998.
- [JP98] Hugues Juille and Jordan B. Pollack. A sampling-based heuristic for tree search applied to grammar induction. In *In Proceedings of AAAI-98*, pages 776–783. MIT Press, 1998.
- [Kas65] Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. AFCRL 65-758, Air Force Cambridge Research Lab, Bedford, M.A., 1965.
- [Kor08] Andras Kornai. *Mathematical Linguistics*. Springer, 2008.
- [KY00] John C. Kieffer and En-Hui Yang. Grammar based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46:2000, 2000.
- [Lan92] Kevin J. Lang. Random DFAs can be approximately learned from sparse uniform examples. In *COLT '92: Proceedings of the fifth annual workshop on Computational Learning Theory*, pages 45–52, New York, NY, USA, 1992. ACM.
- [Lev73] L. A. Levin. Universal sequential search problems. 1973.
- [LPP98] Kevin Lang, Barak Pearlmutter, and Rodney Price. Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm, 1998.
- [LS00] Pat Langley and Sean Stromsten. Learning context-free grammars with a simplicity bias. In *Proceedings of the Eleventh European Conference on Machine Learning*, pages 220–228. Springer-Verlag, 2000.
- [LV91] Ming Li and Paul M. B. Vitányi. Learning simple concepts under simple distributions. *SIAM JOURNAL OF COMPUTING*, 20:911–935, 1991.
- [LV93] M. Li and P. Vitanyi. *An introduction to Kolmogorov complexity and its applications*. Texts and Monographs in Computer Science, Berlin, New York: Springer, 1993.
- [NI00] Katsuhiko Nakamura and Takashi Ishiwata. Synthesizing context free grammars from sample strings based on inductive CYK algorithm. In *Proceedings of the 5th International Colloquium on Grammatical Inference: Algorithms and Applications*, pages 186–195, London, UK, 2000. Springer-Verlag.
- [NLHN09] Jens Nilsson, Welf Löwe, Johan Hall, and Joakim Nivre. Parsing formal languages using natural language parsing techniques. In *Proceedings of the 11th International Conference on Parsing Technologies, IWPT '09*, pages 49–60, Morristown, NJ, USA, 2009. Association for Computational Linguistics.
- [NmW97] Craig G. Nevill-manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm, 1997.
- [OG92] J. Oncina and P. Garcia. Inferring regular languages in polynomial update time.

- In N. Pérez de la Blanca, A. Sanfeliu, and E. Vidal, editors, *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*, pages 49–61. World Scientific, Singapore, 1992.
- [Pap94] Christos M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.
- [PH97] Rajesh Parekh and Vasant Honavar. Learning dfa from simple examples. In *In Proceedings of the Eighth International Workshop on Algorithmic Learning Theory (ALT'97)*, pages 116–131. Springer, 1997.
- [Pit89] Leonard Pitt. Inductive inference, DFAs, and computational complexity. In *AIJ '89: Proceedings of the International Workshop on Analogical and Inductive Inference*, pages 18–44, London, UK, 1989. Springer-Verlag.
- [PPK⁺04] Georgios Petasis, Georgios Paliouras, Vangelis Karkaletsis, Constantine Halatsis, and Constantine D. Spyropoulos. e-GRIDS: Computationally efficient grammatical inference from positive examples. *Grammars*, 2004.
- [PW93] Leonard Pitt and Manfred K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of the Association for Computing Machinery*, 40:95–142, 1993.
- [Sak05] Yasubumi Sakakibara. Grammatical inference in bioinformatics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:1051–1062, 2005.
- [Sch97] Robert R. Schaller. Moore's law: past, present, and future. *IEEE Spectr.*, 34:52–59, June 1997.
- [Sch03] J. Schmidhuber. Gödel machines: self-referential universal problem solvers making provably optimal self-improvements. Technical report, IDSIA, Manno-Lugano, Switzerland, 2003.
- [SHRE05] Zach Solan, David Horn, Eytan Ruppin, and Shimon Edelman. Unsupervised learning of natural languages. *PNAS*, 102(33):11629–11634, 2005.
- [Sim02] Khalil Sima'an. Computational complexity of probabilistic disambiguation. *Grammars*, 5:125–151, 2002. 10.1023/A:1016340700671.
- [SJ03] Marc Sebban and Jean-Christophe Janodet. On state merging in grammatical inference: A statistical approach for dealing with noisy data. In *ICML*, pages 688–695, 2003.
- [Sol64] Ray J. Solomonoff. A formal theory of inductive inference, part I & II. *Information and Control*, 7:1–22, 1964.
- [SSCZ04] Competition Bradford Starkie, Bradford Starkie, François Coste, and Menno Van Zaanen. The Omphalos context-free grammar learning. In *in Y Sakakibara (ed.), Grammatical Inference: Algorithms and Applications; 7th International Colloquium, ICGI 2004*, pages 16–27. Springer, 2004.
- [TB73] B. A. Trakhtenbrot and Ya. M. Barzdin. *Finite Automata, Behavior and Synthesis*. North Holland, Amsterdam, 1973.

- [Tho00] Franck Thollard Thollard. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *In Seventeenth International Conference on Machine Learning*, pages 975–982. Morgan Kauffman, 2000.
- [Val84] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.
- [VB87] Kurt Vanlehn and William Ball. A version space approach to learning context-free grammars. *Mach. Learn.*, 2(1):39–74, 1987.
- [VL00] P.M.B. Vitanyi and Ming Li. Minimum description length induction, Bayesianism, and Kolmogorov complexity. *Information Theory, IEEE Transactions on*, 46(2):446–464, Mar 2000.
- [vS10] Maarten van Someren. The blades of Occam’s razor. to appear, October 2010.
- [vZ00] Menno van Zaanen. Abl: Alignment-based learning, 2000.
- [vZA01] Menno van Zaanen and Pieter Adriaans. Comparing two unsupervised grammar induction systems: Alignment-based learning vs. emile. Technical report, 2001.
- [Wan07] Hui Wang. All common subsequences. In *Proceedings of the 20th international joint conference on Artificial intelligence, IJCAI’07*, pages 635–640, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.
- [WM97] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.
- [You67] Daniel H. Younger. Recognition and parsing of context-free languages in time n^3 . *Information and Control*, 10(2):189 – 208, 1967.
- [Zho07] Lina Zhou. Ontology learning: state of the art and open issues. *Information Technology and Management*, 8:241–252, 2007.