Computing order-independent statistical characteristics of stochastic context-free languages

Larsen, Ole Vilhelm

COMPUTING ORDER-INDEPENDENT STATISTICAL CHARACTERISTICS OF STOCHASTIC CONTEXT-FREE LANGUAGES

Ole Vilhelm Larsen

Laboratory of Image Analysis
Institute of Electronic Systems
Aalborg University
Denmark

Second, revised edition
July 1995

The first edition of this dissertation was submitted in January, 1994 to the Faculty Council of Technical Sciences, Aalborg University, Denmark, in partial fulfillment of the requirements for the Doctor of Philosophy degree. The first edition was approved by the committee, and this second edition is motivated by some minor, mainly typographical revisions.

The Faculty Council appointed the following adjudication committee to evaluate the thesis:

Abstract

Within structural pattern classification the *syntactic* methods form an important subgroup. These methods use formal languages as models for the individual pattern classes. The languages can be grouped according to the complexity of the allowed ordering of symbols. The most used types of languages in pattern recognition and in general are the *context-free* and the *regular* languages. The latter is a subgroup of the former and is computationally simpler. For patterns subject to natural variations or noise *stochastic* languages are often used to provide a statistical description of the variations within the language.

Statistical characteristics of the stochastic languages have a potential use in a pattern recognition system. The main topic of this thesis is to develop methods for computing a set of statistical characteristics being independent of the ordering of symbols in the string for stochastic context-free languages. By focussing on the *order-independent characteristics* it is hypothesised that the computations can be done using the simpler *regular language representation* of the context-free language. Another important topic is to exemplify how such order-independent characteristics can be used within pattern classification.

This dissertation describes the development of the *Parikh-Thomason-Larsen* approach to computing order-independent statistical characteristics of stochastic context-free languages. The characteristics are related to the set of strings in the language or the derivation process of the strings. Since the derivation process of context-free language is significantly dependent on whether the language is *expansive* or *non-expansive*, the approach is formulated differently for the two cases.

For the *non-expansive* case it has been shown that *Pilling's Method* provides a useful tool for obtaining a regular language representation, and that a *generating function* representation of the obtained regular language can be used for the computation of statistical characteristics.

For the *expansive* case methods known from complex function theory have been applied. The *Generalized Lagrange Inversion Formula* has been shown to be applicable to context-free languages in order to obtain a power series representation of the language. From the series order-independent characteristics can be computed.

Various ways in which statistical characteristics can be exploited in pattern recognition have been outlined. To exemplify the potential a *two-stage syntactic classification strategy* has been developed and tested experimentally.

Dansk sammenfatning

Indenfor strukturel mønstergenkendelse udgør de *syntaktiske* metoder en betydningsfuld undergruppe. Disse metoder modellerer de enkelte mønsterklasser ved brug af formelle sprog. Sådanne sprog kan inddeles i et hierarki udfra kompleksiteten af den orden, som symbolerne kan have i forhold til hinanden. *Kontekst-frie* og *regulære* sprog er de to mest anvendte sprogklasser indenfor mønstergenkendelse. Sidstnævnte klasse er en undergruppe af de kontekst-frie og er pga. symbolernes simplere orden mindre komplekse at anvende. Ofte vil mønstre indenfor en klasse udvise naturlige variationer og/eller være påvirket af støj. I sådanne situationer kan klassifikationen med fordel gøre brug af *stokastiske* sprog, som giver mulighed for at etablere en statistisk beskrivelse of sprogets variationer.

Statistiske karakteristika for stokastiske sprog har potentielle anvendelser indenfor mønstergenkendelse. Hovedtemaet for denne afhandling er udvikling af metoder til beregning af statistiske karakteristika for stokastiske kontekst-frie sprog. Der fokuseres udelukkende på karakteristika, der ikke er påvirket af den orden, i hvilken symbolerne indgår. Derved opnås, at de egenskaber ved det kontekst-frie sprog, som er væsentlige for karakteristikaene, kan modelleres som et simplere regulært sprog. Et andet væsentligt tema er belysningen af, på hvilken måde de beregnede karakteristika kan udnyttes indenfor mønstergenkendelse.

Denne afhandling beskriver udviklingen af *Parikh-Thomason-Larsen* metoden til beregning af ordensuafhængige statistiske karakteristika for stokastiske kontekst-frie sprog. Karakteristikaene vedrører strengene i sproget og deres afledningsproces. Sidstnævnte er stærkt afhængig af, om sproget er *ekspansivt* eller *non-ekspansivt*, hvilket har givet anledning til, at metoden er formuleret forskelligt for de to situationer. For de *non-ekspansive* kontekst-frie sprog er det blevet vist, at *Pillings metode* er brugbar til at repræsentere sproget vha. et regulært sprog samt, at *frembringende funktioner* kan anvendes til at beregne statistiske karakteristika udfra det regulære sprog. For de *ekspansive* kontekst-frie sprog er der blevet gjort brug af teknikker kendt fra kompleks funktions teori. Den generaliserede udgave af *Lagranges Inversions Formel* danner grundlag for, at sproget repræsenteres som en potensrække, udfra hvilken karakteristika kan uddrages.

I afhandlingen er der ydermere en overordnet beskrivelse af, hvorledes statistiske karakteristika for formelle sprog kan udnyttes indenfor mønstergenkendelse. For at understøtte denne er der udviklet en *to-trins klassifikationsstrategi*, som er

blevet eksperimentelt verificeret.

Acknowledgements

This thesis is the result of work carried out at the Laboratory of Image Analysis at Aalborg University.

A number of individuals made this research possible and I wish to express my sincere thanks to all of them.

I am indebted to my supervisor Professor Erik Granum for many interesting ideas and proposals in connection to my project. He is furthermore responsible for creating the stimulating environment which has made this research possible.

A visit to the Computer Science Department at the University of Tennessee, Knoxwille, as a Research Scholar made it possible for six months to work under the supervision of Professor Michael G. Thomason. The ideas from which this thesis have grown were initially put forward by him. I am deeply grateful for his invaluable guidance and support during the initial phase but also throughout the project.

Thanks are also due to my family for their strong support and encouragement throughout my studies. Britt and Patrick deserve a special thanking for their almost unlimited patience and tolerance when I was deeply involved in details of my work.

Finally, the Danish Research Academy is gratefully acknowledged for the funding that made my stay at the University of Tennessee, Knoxwille, possible.

Contents

Chapter 1

Introduction and Overview

Ever since the early days of modern computer technology there has been a significant interest in emulating the intelligent behaviour of human beings by computers. Such behavior involves human perception of the environment, which includes the ability to recognise and perceive various 1, 2 and, 3 dimensional patterns. Especially the perception of acoustic and visual patterns have attracted significant interest in the past 2 decades.

Any pattern recognition system can be in one of two possible modes of operation: i) *Analysis mode* (learning), and ii) *Recognition mode* (application). In the analysis mode test patterns are analysed in order to learn which information is relevant for a classification. In the recognition mode new patterns are classified according to the knowledge obtained in the analysis mode. When the analysis mode has been executed once on sets of learning patterns, the recognition mode can be applied to new patterns as often as required.

The techniques for computer based pattern recognition can be grouped into 4 groups, i) statistical, ii) structural, iii) artificial intelligence based, and iiii) artificial neural networks.

Statistical methods are based on a pattern description where each sample is represented by a n-dimensional feature vector. Each feature is a numerical measure of a characteristic of the sample, e.g., its length or colour. During the analysis mode the features are chosen, and the statistical variations within each class are described and evaluated in an attempt to partition the entire feature space into mutually exclusive regions, with each region belonging to a specific pattern class. Recognition is the process, where it is determined in which region the unknown sample falls and subsequent to which pattern class it belongs.

Structural methods make explicit use of knowledge about the structure of the object. It utilize a pattern description where each pattern is divided into sub-patterns called *primitives*. Each primitive has no direct relation to the structure of the pattern. A pattern is represented by knowledge about how sub-patterns must be combined to make up the entire pattern, and how sub-patterns relate to each other. The structural techniques can be further subdivided into structural prototypes and *syntactic methods*. Only the latter will be described in more detail. For the syntactic methods the analysis mode consists of constructing rules for combining primitives in order to obtain the structure of a given object. The methods are formulated around the concept of *formal languages* with each primitive represented as a terminal symbol and a grammar inferred for each pattern class. The recognition mode consists of checking whether an object could be constructed using the rules associated with a specific object class. This process is called the *syntax analysis* or the *parsing* of a string.

In methods based on *artificial intelligence*, numerical, structural, and other types of information are used to describe the pattern. The analysis mode consists of, determining the appropriate discriminant information, specifying each pattern class as a kind of abstract concept, and specifying the conceptual relations between the entities involved in the system. The recognition phase involves taking observations relating to an object and using an inference engine to determine whether or not a pattern can be a specific instance of an abstract concept.

The *artificial neural net* approach has many similarities with statistical pattern recognition concerning both the data representation and the classification principles. The practical implementation is, however, very different. The analysis mode involves the configuration of a network of artificial neurons and the training of the net to determine how the individual neurons can effect each other. The recognition mode involves sending data through the net and evaluating which class got the highest score.

Further to the above mentioned "pure" methods, various *hybrid methods* have been designed to bring together the power of two or more of the pure methods.

The syntactic pattern recognition methods make up the framework for this thesis.

1.1 Aspects of syntactic pattern recognition

An important problem for the syntactic methods is their sensitivity to variations in data. In most real applications variations in the data will exist. This can be due to natural variations in the data itself, to the acquisition process required for the computerization of the recognition process, or to the extraction of primitives. The problem is that the language representing a pattern class inferred from a set of learning samples is not affected by the statistics of the variations, e.g., one hundred identical strings close to the ideal version of the object will not add more information concerning the rewriting rules than one sample with significant variation in relation to the ideal representation. This is a significant difference to statistical methods where such variations are modelled explicitly. If variations in data result in large pattern classes the problem of overlapping classes might occur. In such cases there is no way to measure whether the classes overlap because of some wild outliers or because the classes have some significant similarities.

Stochastic languages have been introduced as a means of modelling the variations in data. A probability is attached to each rewriting rule denoting how likely it is to be used in a generating process of strings in the language. Rewritings due to outliers will therefore be given a low probability. Based on the production probability a probability of the existence of each string in the language can be computed. Strings caused by outliers will get a relative lower probability than strings close to the typical pattern. In case of overlapping classes it is possible to assign the pattern to the class most likely to have generated it.

Significant work on how to use the statistical information for pattern recognition purposes has been reported. Three major areas can be defined: i) Description of pattern variations, [5, 9, 13, 12, 19, 16, 17, 21, 20, 51], ii) Classification strategies for overlapping classes, [7, 9, 19, 17, 31, 32, 51] and iii) Control of the syntax analysis of strings, [31, 34, 43, 37, 47]. The majority of papers has focussed on the use of the production probabilities and the string probability derivable thereof, and only few have used other statistical characteristics.

Stochastic languages serve as a mechanism for getting statistical information into the structural model, and it thereby makes it possible to compute a variety of *statistical characteristics* related to the strings and to the grammar which describe the process of generating the strings, e.g., mean string length or mean number of occurrences of a specific primitive.

The methods for computing statistical information are based on mathematical models of the process which create the individual patterns within a pattern class. Properties of the model can then be given an interpretation in terms of characteristics of the pattern class, thus without ever generating a single pattern, knowledge about the entire pattern class can be obtained. It is primarily the mean values of random variables for the entire class which are computable, e.g., the mean number of symbols in a symbolic representation of all the patterns. One of the models does, however, make a controlled enumeration of all strings in the language possible, allowing for characteristics other than the mean values to be computed.

One of the major advantages of syntactic pattern recognition is that it can make use of the large theoretical and mathematical foundation of formal language theory. Formal languages are grouped into a hierarchy, Chomsky's hierarchy, consisting of 4 levels: i) unrestricted , ii) context-sensitive, iii) *context-free* and iiii) *regular* languages. Regular languages form a subgroup of the context-free which in turn is a subgroup of the context-sensitive language etc..

Choosing a specific language type for a given recognition problem is always a trade-off between the descriptive power of the type, the computational complexity of the algorithms associated with it, and the amount of properties precisely defined for the type. The first two decrease moving down the hierarchy, the latter increases. Context-free languages are widely used, because they are sufficiently restricted to allow for a precise definition of many properties, the recognition can be done in low-order polynomial time, and they are powerful enough to allow for real world applications. Many applications within Natural language processing and Programming languages have been reported. Also most pattern recognition problems addressed by the syntactic approach have been modelled by regular or context-free languages.

1.2 The motivation and aim of the study

Statistical characteristics contain information which has a potential use in a pattern recognition system. The characteristics related to the primitives are relevant for the classification performance. The set of characteristics of a pattern which contains enough discriminative power to separate the classes will not be known prior to the analysis mode. Therefore all possible characteristics including the statistical must be processed. Clearly it is of interest to know the mean number of primitives in a pattern, or to know whether or not one primitive is more likely than another.

Characteristics related to the derivation process will not directly effect the error-rate of the classification, but can play a role in designing the model/grammar so it most effectively expresses the language of interest. As an example it is of interest to know the mean number of rewritings required to generate a string from a given class. This is important since during classification it is necessary to establish which productions were used to derive the string. The processing time will therefore be an increasing function of the number of productions used.

Statistical characteristics not related to the ordering of the symbols of the string form an important subset of characteristics; all those mentioned so far have been based on order-independent random variables.

Often the high complexity of context-free languages are required in pattern recognition situations. Such languages are conceptually more difficult than regular languages as the ordering of symbols are more complex. Any model for context-free languages used for computing statistical information will therefore always be more complex than its regular counterpart. From formal language theory it is known according to Parikh [35] that it is possible for non-stochastic languages to represent the information not related to the ordering of symbols by a simple model, namely a regular language.

It is the primary goal for this study to determine, if the regular language representation for non-stochastic languages can be extended to stochastic languages and if so, which statistical characteristics are computable from the simpler model.

It is the working hypothesis that:

> *Statistical characteristics related to order-independent random variables for stochastic context-free languages can be computed utilizing a regular language representation.*

A secondary goal is to gain new insight in how statistical characteristics can be utilized within the field of syntactic pattern recognition.

1.3 An outline of the dissertation

The thesis comprises 7 chapters and 2 appendices.
Chapter 2 introduces the concept of stochastic languages, and outlines the mathematical tools for manipulating such languages. It establishes the general notation and terminology used in successive chapters. Finally, a set of characteristics is described and it is shown how expansiveness of a grammar significantly effects the derivation-process for strings in the language. The last observations result in the design of two separate methods for the computation of characteristics.
Chapter 3 presents a new method for computing order-independent random variables for non-expansive stochastic context-free languages and proves why a direct extension to the expansive case is not possible.
Chapter 4 presents a method for computing the variables for expansive stochastic context-free languages.
Chapter 5 gives an overview of the applicability of the existing and the new methods to computations of specific characteristics.
Chapter 6 is devoted to an application in which statistical characteristics are used in a syntactic pattern recognition system to allow for flexible processing time.
Chapter 7 summarises and concludes the results presented in the thesis and points in the direction of future work.
Appendices A and B describe in detail how Markov chains and multi-type branching processes can be used to compute statistical information about regular and context-free languages respectively. Strong emphasis has been given to an algorithmic description complementing the traditional description of the mathematics of the models.

Chapter 2

An Introduction to Stochastic Languages

As mentioned in the previous chapter one of the advantages of syntactic pattern recognition is that it can make use of the large theoretical foundation of formal language theory. This chapter gives a very brief review of relevant parts of formal language theory used in this study. Through a set of definitions and theorems it establishes the general notation and terminology used in successive chapters. The main emphasis is put on *stochastic languages* and their representations in terms of *stochastic grammars*.

Finally the set of characteristics are described, and it is shown how expansiveness of a grammar significantly effects the derivation-process for strings in the language.

2.1 Formal languages

Let there be given an alphabet $A = \{a, b, c, ..., z\}$ of finite cardinality and let $A^{\star}$ denote all strings of finite length which can be build using elements from A. A formal language L is any finite or countable infinite subset of $A^{\star}$. Any formal language can be defined either by i) a listing of all strings in the language, either explicitly or implicitly using a set-notation, ii) specifying an abstract machine called an automaton capable of recognizing all strings in L and nothing else, or iii) specifying a mathematical model called a grammar capable of generating all strings in the language and nothing else.

In this thesis the languages will be defined in terms of grammars.

A grammar can formally be specified as a 4-tuple $G = (N, \Sigma, P, S)$ where N and Σ are finite sets of symbols called nonterminals and terminals, respectively (Σ is identical to the alphabet of the language). S is a unique starting symbol, and P is a finite set of productions or rewriting rules with the general form $\alpha \to \beta$ where α and β are in $(N \cup \Sigma)^{\star}$.

By imposing restrictions on how rewriting rules can be composed it is possible to partition formal languages into a 4 level hierarchy called *Chomsky's hierarchy*,

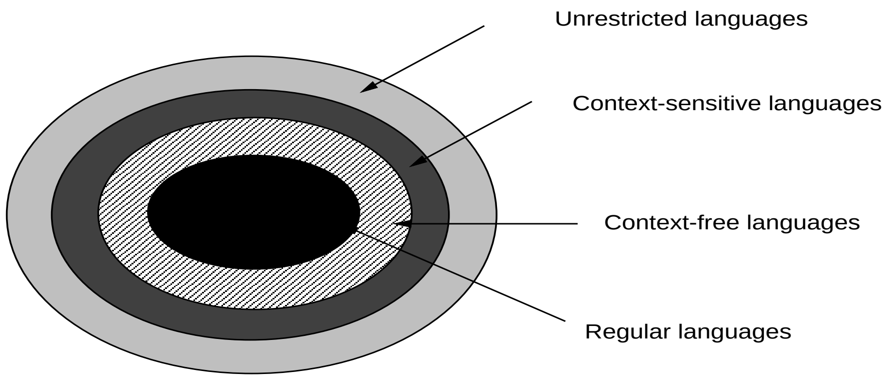


Figure 2.1: Chomsky's hierarchy of formal languages.

as shown in Figure 2.1. It is called a hierarchy since the regular are subsets of the context-free languages which in turn are subsets of the context-sensitive etc. In this thesis only context-free and regular languages are of interest and their rewritings have the following characteristic forms;

Regular grammars:	$A \rightarrow aA$
	$A \rightarrow b$
	for $A \in N$ and a, $b \in \Sigma$.
Context-free grammars:	$A \rightarrow \beta$
	for $A \in N$ and $\beta \in (N \cup \Sigma)^\star$

The difference between regular and context-free productions is that the right-hand side can consist of any number of terminals and nonterminals for a context-free production, whereas it is limited to at most one nonterminal and one terminal for regular productions. From this it becomes apparent that the ordering of symbols in a context-free language can be much more complex than in regular languages.

Define two relations, $\Rightarrow$ and $\overset{\star}{\Rightarrow}$ on $(N \cup \Sigma)^\star$. If ρ and δ are strings in $(N \cup \Sigma)^\star$ then we use the notation $\rho\alpha\delta \Rightarrow \rho\beta\delta$ to indicate that $\rho\beta\delta$ can be *derived* from $\rho\alpha\delta$ by using a single production, $\alpha \rightarrow \beta$. If $\rho_1, \rho_2, \rho_3, \cdots, \rho_m$ are strings in $(N \cup \Sigma)^\star$, $m \geq 1$ and

$$\rho_1 \Rightarrow \rho_2, \rho_2 \Rightarrow \rho_3, \cdots, \rho_{m-1} \Rightarrow \rho_m$$

then we use $\rho_1 \overset{\star}{\Rightarrow} \rho_m$ to denote that zero, one, or multiple rewritings must be used to derive ρ_m from ρ_1. The sequence $\rho_1, \rho_2, \cdots, \rho_m$ is called a *derivation.* The language $L(G)$, generated by grammar G, is the set $\{w \mid w \in \Sigma^\star \wedge S \overset{\star}{\Rightarrow} w\}$. Strings must only consist of terminals and be derivable from the starting symbol. A string α of terminals and nonterminals is called a *sentential form* if $S \overset{\star}{\Rightarrow} \alpha$.

There exist a so-called *pumping lemma* for regular and for context-free languages, e.g., see Aho and Ullman [1].

For regular languages it states: Let L be a regular language. A constant k exist such that if a string w is in L and $|w| \geq k$, then w can be written as xyz, where $0 < |y| \leq k$ and $xy^iz \in L$ for all $i \geq 0$.
For context-free languages it states: Let L be a context-free language. A constant k exist such that if $|z| \geq k$ and $z \in L$, then z can be written as $uvwxy$ where $|vx| \geq 1$, $|vwx| \leq k$, and for all $i > 0$, uv^iwx^iy is in L.
The lemmas state that for strings above a certain length the derivation process will contain a loop which will give rise to the repetition of a substring within the string. Since there is no mechanism for controlling how many times a derivation goes through a loop, any string resulting from any number of runs through the loop must be considered a part of the language.

Often it is useful to present a derivation graphically, as done extensively in this thesis. This is made possible by using *derivation trees* defined by K.S. Fu [18] as:

Definition 1 *A labeled ordered tree D is a* **derivation tree** *for a context-free grammar $G = (N, \Sigma, P, S)$ if:*

1. *Every node of the tree has a label, which is a symbol in $(N \cup \Sigma)$.*
2. *The root of the tree has the label S.*
3. *If a node has at least one descendant other than itself, and has the label A, then $A \in N$.*
4. *If nodes $n_1, n_2, \ldots, n_m$ are the direct descendants of node n(with label A) in the order from the left, with labels $A_1, \ldots, A_m$, respectively, then*

$$A \rightarrow A_1 A_2 \ldots A_m$$

   must be a production in P.

As an example of derivation trees Figure 2.2 shows the trees for the two strings *aba* and *ababa* generated with a grammar having the productions; $S \rightarrow aAS$, $S \rightarrow a$, $A \rightarrow bSA$, and $A \rightarrow b$. To every derivation of a string in a context-free language there corresponds a derivation tree and vice versa.

In some cases there are different ways to derive a string. A grammar is then called an *ambiguous* grammar. The formal definition follows Aho [1];

Definition 2 *A context-free grammar G is said to be* **ambiguous** *if there is at least one string* $\mathbf{w}$ *in $L(G)$ for which there is more than one distinct derivation tree with frontier* $\mathbf{w}$.
Two derivation trees are different if the left to right ordering of nodes at any level of the tree is different.

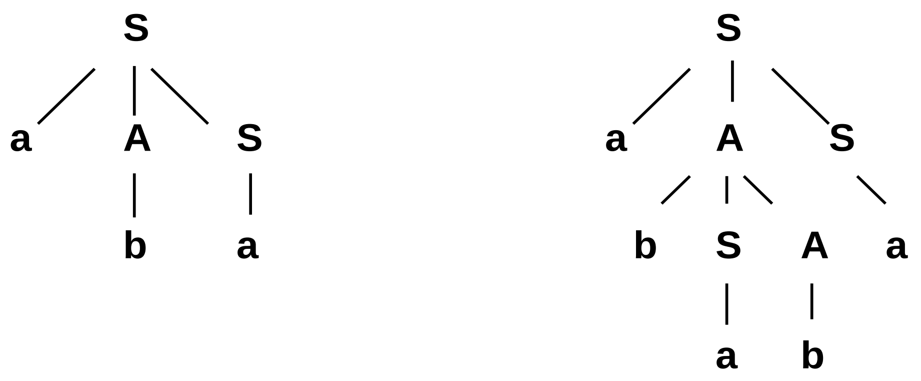


Figure 2.2: Two derivation trees for the strings *aba* and *ababa*.

2.2 Stochastic Languages

A stochastic language is a collection of strings with a probability distribution defined on the set of strings. The formal definition follows Booth [6],

Definition 3 A stochastic language *over the alphabet* Σ *is a pair (L,p), where* p *is a function* $p : \Sigma^* \rightarrow \mathbf{R}$. *Three restrictions are defined on* p[1]*;*

1. $p(x) = 0$ *for* $x \notin L$.
2. $0 < p(x) \leq 1$ *for* $x \in L$.
3. $\sum_{x \in L} p(x) = 1$.

For a stochastic language it is required that the sum of weights for all strings equals 1, making p a probability function. Stochastic languages with strings which has no upper limit on their length constitute a special situation since the strings will have a *very* small weight as stated by Wetherell [49] in the following theorem:

Theorem 1 *Let (L,p) be a stochastic language. Then for each* $\epsilon > 0$ *there is a number* $N \geq 0$ *such that*

$$|x| \geq N \Rightarrow \epsilon > p(x)$$

where $|x|$ *denotes the length of string x.*

For infinite languages it is not feasible to check whether the sum of probabilities equals one, simply by adding the weights, since there will always be strings of greater length representing a small part of the total probability mass. In such a situation it is required to obtain a model for the generation process and perform a test equivalent to evaluate its behaviour in the limit as the processing time goes to infinity [25, 6].

[1]The second restriction differs from the formulation by Booth concerning the lower limit of $p(x)$. Booth defines it as $0 \leq p(x) \leq 1$ for $x \in L$.

Stochastic languages are typically specified by a *stochastic automaton* or a *stochastic grammar*. For a given stochastic language there is a one-to-one correspondence between the strings accepted by the automaton and the strings generated by the grammar meaning that both models can accept/generate the same strings with identical probabilities. Any string in a language can be modelled correctly concerning its syntactic structure by the corresponding automata/grammar but probabilities might be assigned to a string in such a way that none of the models can reproduce strings with the correct probability as stated in the following theorem by Booth [6]:

Theorem 2 *There are stochastic languages for which no stochastic grammar or stochastic automata can reproduce the correct probability for all the individual strings in the language.*

There are several explanations for this. As stated through the pumping lemmas loops in the derivation process can result in strings differing from each other only on the number of repetitions of a substring. The probabilities of such strings will be dependent on each other, and they therefore cannot take any arbitrarily probability. Furthermore a string is the result of a sequence of derivation steps, each with a predefined fixed probability. Therefore the probabilities assigned to generated strings, come from a discrete set of probabilities, and this fact can conflict with the fact that a string according to Definition 3 can be assigned *any* value in the range from 0 to 1.

Due to the inference mechanism creating the grammar the problem arising from Theorem 2 is of no real concern when using stochastic languages for syntactic pattern recognition, and most other applications. The grammar is typically inferred from a finite set of learning samples, in such a way that it is capable of generating a set of strings of greater cardinality (possible countable infinite) having identical syntactic structure. In such a situation it is not relevant to maintain the exact probability of individual strings in the learning samples.

2.3 Stochastic Grammars

A stochastic grammar is a 4-tuple $G_s = (N, \Sigma, P, S)$ where N and Σ are finite sets of symbols. S is a unique starting symbol and P is a finite set of productions with the general form

$$p_i : \alpha \rightarrow \beta$$

where α and β are in $(N \cup \Sigma)^\star$. p_i is the probability that α is rewritten as β in the derivation process for all elements in the language generated by G_s, $L(G_s)$.

If the productions in G_s are stripped for the probabilities, the result is a grammar called the *characteristic* grammar of G_s. The characteristic grammar can be classified according to Chomsky's hierarchy. A stochastic grammar is named after the place in Chomsky's hierarchy obtained by its characteristic grammar, e.g., a stochastic context-free grammar (SCFG) generates a stochastic context-free language (SCFL).

The stochastic grammar serves as a stochastic generating mechanism for strings in a language. To be stochastic every step in the derivation process must be described by a probability function. This means that a probability function must control which of the possible production to use when a given nonterminal is to be rewritten. This leads to the following definition;

Definition 4 *Let there be given a grammar G = (N,Σ,P,S) in which N consist of n nonterminals. The set of productions can be partitioned in n equivalence classes $C_1, C_2 \ldots, C_n$, each consisting of productions having the same nonterminal as the left-hand side. A grammar is* **proper** *if*

$$\sum_{p_i \in C_j} p_i = 1$$

This thesis will concern only proper grammars.

Assuming that the use of one production is independent of which productions were used earlier in the derivation, a probability for the entire string can be computed as the product of the individual productions involved.

Definition 5 *Given a derivation $\Delta =< i_1, \ldots, i_n >$ then the* **probability, p(Δ) of a derivation Δ** *is*

$$p(\Delta) = \prod_{1 \leq k \leq n} p_k$$

where p_k is the probability of the kth production in Δ.

If a string is ambiguous and therefore can be derived in different ways, the probability is the sum of probabilities for the individual derivations.

Definition 6 *Given a string of symbols $x = (a_1 a_2 \ldots a_n)$. Let $\{\Delta_i\}$ for $1 \leq i \leq m$ be the set of derivations giving x, then the* **probability, p(x) of a string** *is*

$$p(x) = \sum_{1 \leq i \leq m} p(\Delta_i)$$

For the language induced by the grammar to be probabilistic it is required that the grammar must be proper and consistent.

Definition 7 *Let there be given a grammar G= (N,Σ,P,S) and let L(G) denote the set of strings generated by G then if,,*

$$\sum_{x \in L} p(x) = 1$$

the grammar is said to be **consistent.**

There exist methods to test consistency for regular and context-free stochastic grammars. For detail see Appendix A and B.

2.4 Characteristics

Strictly speaking the term "characteristic of a stochastic language" only covers the information related to strings, as a sequence of primitives, in the language. There is, however, a very tight connection between the language and the grammar generating strings in the language. Therefore information related to the grammar and the derivation process described by the grammar will be considered characteristics of the language in this thesis.

Characteristics can be grouped into the i) *non-statistical* and the ii) *statistical* characteristics. The first group covers the information which is available without the process of counting, e.g., the Chomsky language type, the set of nonterminals, the set of terminals. The statistical characteristics are by far the largest group and cover everything which can be asked for by the questions "How many ...?" and "How often ...?". Each characteristic is related to a random variable, e.g., a random variable can be the number of occurrences of all terminals, and based on that the mean string length can be computed and used as a characteristic of the language.

In general it is of interest to compute 3 things for each random variable:

- Probability of the random variable taking a specific value, e.g., $P(x = 4)$.
- Probability of a random variable being in a given interval, e.g., $P(2 \leq x \leq 4)$.
- Mean and variance of the random variable.

The random variables can be further divided into i) *order-dependent* and ii) *order-independent* random variables. If it is associated with a terminal having a specific position in the string or with a specific ordering of two or more terminals in the string it is order-dependent. Likewise for random variables related to the nonterminals and the productions. If the variable is only related to how often things occur independently of where in the string or in the derivation it happens it is order-independent. Formulated in terms of terminals, nonterminals, and productions the following list outlines the possible groups of order-independent random variables for stochastic languages.

- The number of occurrences of a specific terminal,
- The number of occurrences of a group of terminals,
- The number of occurrences of a specific nonterminal,
- The number of occurrences of a group of nonterminals,
- The number of applications of a specific production,
- The number of applications of a group of productions,

The terms *single* and *multiple entity random variable* will be used to denote if the variable only relates to just one terminal, one nonterminal, or one production or if it involves two or more. The number of occurrences of terminal a is a single entity variable, whereas the number of joint occurrences of a's, b's and c's is a multiple entity variable.

As previously indicated the random variables can either be related to the strings in the language in which case the variables are based on the terminals or be related to the grammar and the derivation process based on nonterminals and productions.

The grouping of random variables for stochastic languages can be summarized in the following graph

Since the grammar is a generative device it can be used for a probabilistic enumeration of strings driven by a random-number generator. By computing frequency counts on the produced strings it would be possible to compute statistical characteristics for the language. This approach does, however, come up against two major problems; how many strings should the grammar generate in order for a sufficient number of different strings to be available for confident estimation of the probabilities, and how to ensure that the random-number source controlling the derivation process is unbiased. Any bias would directly effect the characteristics.

Another approach would be to use a mathematical model of the derivation process. The probabilistic information about the production rules of the grammar can be converted to information for the model. Statistical properties computed for the model can then be interpreted as characteristics of the derivation process. As described in Chapter 5 and in Appendices A and B, *Markov chains* can be used to model the derivation process of stochastic regular languages, whereas the theory of *multi-type branching processes* must be used to deal with the high complexity of stochastic context-free languages.

2.5 Expansive grammars

The concept of *expansive grammars* play a special role in this thesis.

Definition 8 *A grammar* $G=(N,\Sigma,P,S)$ *is* **expansive** *if a derivation in* G *exist where*

$$A \overset{*}{\Rightarrow} \alpha A \beta A \gamma$$

for α, β, γ *in* $(N \cup \Sigma)^{\star}$*, and* A *in* N*.*

Whether a language is expansive or not cannot always be detected by inspection of the productions. It is a characteristic of the derivation process, but it can always be determined. An algorithm for detection of expansiveness in a context-free grammars is given by Baron and Kuich [3].

	0	1	2	3	4	5	6	7	8	9
G_1	1	1	1	1	1	1	1	1	1	1
G_2	1	1	2	5	14	42	132	429	1430	4862
G_3	1	1	3	12	55	273	1428	7752	43263	246675

Table 2.1: Number of different strings having the same number of occurrences of symbols for 3 grammars having different productions. The productions introducing 1, 2, and 3 S's have been used 0 to 9 times each.

The concept of expansive grammars is not specifically related to stochastic languages, but it is a concept uniquely linked with the complexity of the generating process for the strings. If a nonterminal can reintroduce two occurrences of itself in the derivation process, it contains a mechanism by which the number of nonterminals in a sentential form can grow in an unbounded way. As an illustration of this consider 3 almost identical grammars $G_i = (N, \Sigma, P, S)$, $N = \{S\}$, $\Sigma = \{a, b\}$. The only difference between the grammars is the productions.

$$\begin{array}{lll} G_1: & S \rightarrow aS, & S \rightarrow b \\ G_2: & S \rightarrow aSS, & S \rightarrow b \\ G_3: & S \rightarrow aSSS, & S \rightarrow b \end{array}$$

G_2 and G_3 are clearly expansive, whereas G_1 is non-expansive. For order-independent random variables the ordering of symbols is not important, and all different strings which have an identical number of symbols add to the value of the random variable. Table 2.1 shows the number of such strings for varying number of applications of the productions $S \rightarrow aS$, $S \rightarrow aSS$, and $S \rightarrow aSSS$ in the respective grammars. It clearly shows the very rapid growth in strings having identical number of symbols for the expansive grammars compared with the non-expansive grammars and the dramatical effect it has when three instead of two occurrences are introduced for each rewriting.

The rapid growth is very important, when a regular language representation of the context-free language is to be constructed. Due to the significant differences in the derivation process for expansive and non-expansive context-free grammars it is advantageous to verify the working hypothesis of this thesis by designing 2 different methods each having just the complexity needed for the situation. The two methods is described in Chapter 3 and 4.

Chapter 3

A method for non-expansive stochastic context-free languages

The work described in this thesis aims at determining whether a regular language representation can be used for computing order-independent characteristics for stochastic context-free languages. In Chapter 2 it was shown that the derivation process for strings for non-expansive and expansive grammars is very different. It has therefore been suggested that two methods, one for each type of the derivation process would be required. This chapter develops a *new method applicable to non-expansive stochastic context-free grammars*, i.e., the first of the two cases. The method is proposed as an hypothesis, and the following sections are devoted to verifying the hypothesis and describing how statistical order-independent characteristics can be computed using the method. At the end of the chapter it is shown why the method is not directly expandable to the expansive case.

3.1 The hypothesis

Strings in a context-free language can have a complicated ordering of symbols. Any model capturing this ordering will be inherently complex. If the information of interest is not related to the ordering of symbols, as is the case for order-independent characteristics, it may be possible to make use of a simpler language representation. The fundamental idea behind the method is that as the ordering of symbols has no relevance to order-independent characteristics a representation in which the symbols are allowed to commute will not put constraints on the computability of the characteristics. This is of significant interest, since Parikh [35] has shown that a regular language representation can indeed be proven to exist for a context-free language in which the nonterminals and the terminals are allowed to move around and take any position in the strings and their intermediate sentential forms. This statement is only proven for non-stochastic languages. Hence the use of it, in the context of stochastic languages, requires a change in domain from stochastic to non-stochastic languages. This may be done by means of a derivation language. In the new domain the production probabilities play the role of unique labels and as the only terminals. Based on the context-free derivation grammar a regular language representation is obtained. Each string in this new language represents a derivation in the original stochastic language. As the terminals of the string are production labels every statistical knowledge related to the stochastic domain can be reintroduced by a proper interpretation of the strings in the non-stochastic domain.

The above mentioned approach can be stated in a more precise manner by the following hypothesis[1]:

> *Order-independent characteristics for stochastic non-expansive context-free languages can be computed using the following three-step-procedure:*
>
> 1. *Change the stochastic context-free grammar into a non-stochastic context-free derivation grammar.*
> 2. *Represent the derivation grammar as a set of regular expressions and solve the set of expression equations.*
> 3. *Represent the solution of a set of regular expression equations by means of generating functions.*

The following sections will develop the hypothesis step-by-step and verify its validity. At the end of the chapter it is shown why this method is limited to the non-expansive case.

3.2 Step one

In an attempt to utilize existing knowledge and techniques known for non-stochastic languages it is of interest to make a transformation of the problem of obtaining the regular language representation of a context-free language from the stochastic domain to the non-stochastic domain. The minimal information about the stochastic grammar needed in the non-stochastic domain is that of the syntactic structure of the productions, and which specific stochastic production relates to a given non-stochastic production. The syntactic structure is determined entirely by the configuration of the nonterminals in the productions. For the two domains the productions must be in a one-to-one correspondence with each other.

The above ideas of a transformation are exactly those embodied in the concept of derivation languages. The definition follows from Altman & Banerji [2].

Definition 9 *Let L be any context-free language. The* **derivation language DL(L)** *is defined as the set of strings for which there is a one-to-one correspondence between the string and a derivation in L. A grammar DG capable of generating all strings in DL is called a* **derivation grammar***.*

[1]This hypothesis was initially put forward by Professor Michael G. Thomason, Department of Computer Science, University of Tennessee, Knoxville, Tennessee, USA.

ALGORITHM 1. Construct a derivation grammar from a SCFG.

Input:

Output:

Method:

A stochastic context-free grammar $G = (N, \Sigma, P, S)$.

A context-free derivation grammar $DG = (N', \Sigma', P', S')$.

1) $N' = N, S' = S$

2) Σ' = $\{p_i\}$, where p_i is the production probability for productions in P, here used as a unique label.

3) Change a production from G having the form

$$p_i : A_i \rightarrow x_1 A_1 x_2 A_2 \cdots x_n A_n$$

where $x_i \in \Sigma$ and $A_i \in N$, into a production in DG on the form;

$$A_i \rightarrow p_i A_1 A_2 \cdots A_n$$

In order to create a derivation grammar for a context-free grammar it is necessary to assign a unique label to each production in the context-free grammar. Then a new grammar is constructed having the same set of nonterminals, but the terminals are being replaced by the set of unique production labels and each production is altered to contain only one terminal, namely the unique label for the production that it is associated with.

The work described in this thesis utilizes that the production probabilities of a stochastic production can be interpreted as a unique label instead of as a numerical value. The detail about the transformation into a derivation grammar is described in Algorithm 1.

As the information about the nonterminals is maintained, the syntactic structure of the strings generated by the two different grammars will be alike. Since the only terminals in a derivation grammar are production labels, there will be a one-to-one correspondence between a string generated by a derivation grammar and a sequence of productions in the original grammar.

Example 1:
Let two grammars G and DG be given, so that DG is a derivation grammar for G.
$G = (\{S, A\}, \{a, b\}, P, S)$ where P contains the productions,
$p_1 : S \rightarrow aSA$, $p_2 : S \rightarrow b$, $p_3 : A \rightarrow bA$, $p_4 : A \rightarrow a$.

$DG = (\{S, A\}, \{p_1, p_2, p_3, p_4\}, P', S)$ where P' contains the productions,
$S \to p_1SA$, $S \to p_2$, $A \to p_3A$, $A \to p_4$.
Consider the following two derivations;

$$S \overset{p_1}{\Rightarrow} aSA \overset{p_2}{\Rightarrow} abA \overset{p_3}{\Rightarrow} abbA \overset{p_4}{\Rightarrow} abba$$

$$S \Rightarrow p_1SA \Rightarrow p_1p_2A \Rightarrow p_1p_2p_3A \Rightarrow p_1p_2p_3p_4$$

The string $p_1p_2p_3p_4$ in $L(DG)$ represents the derivation sequence required to derive the string $abba$ in $L(G)$.

It is important to note that a stochastic context-free grammar gives rise to a context-free derivation grammar, so in this thesis it only serves as a mechanism to change a stochastic grammar into a non-stochastic one.

3.3 Step two

A regular language representation is now to be obtained. Two different approaches have been advised in formal language theory with one thing in common that no representation can be found which will give a full representation of every aspect of the languages and will at the same time be applicable to the entire class of context-free languages. This is, of course, straight forward since otherwise Chomsky's hierarchy would be meaningless. One has to choose between:

- Full representation of all aspects of a language but only applicable to a subset of the context-free languages.
- A representation of a subset of aspects but applicable to all context-free languages.

Altman & Banerji [2] have developed a representation limited to the class of non-expansive grammars. For that subset of the context-free languages a complete representation can be obtained. The basic idea is that the number of nonterminals used in the sentential forms of a derivation for any string in a non-expansive language will have an upper bound. Then the total number of arrangements for these nonterminals for a given sentential form will be finite. If each of the arrangements are represented as a state in a finite state machine, then the transition between states corresponding to the correct arrangements will describe a derivation of a string in the context-free language. Since the machine is a finite state machine a regular language representation has been obtained.

Parikh [35] has taken the second approach and proved that any aspect except those relating to the ordering of the symbols can have a regular language representation and that it can be applied to all context-free languages.

Despite the fact that the chapter is concerned with the design of a method for non-expansive grammars, the approach taken by Parikh will be the one pursued in this work. The reason for this apparent paradox is that Altman & Banerji's method requires a considerable augmentation of the context-free grammar to make it regular in order to model the ordering of the symbols. Since that ordering is of no interest for the computation of order-independent random variables it provides no real and practical alternative to Parikh's approach which need no augmentation of the grammar and carries no knowledge about the ordering of symbols.

3.3.1 Parikh's Theorem

To give a more formal description of Parikh's theorem it is required to define terms like *commutative map, letter-equivalence*, and *semi-linear sets* which will be given below.

Let J denote the non-negative integers and let J^n be a set of n-element vectors, where each element is from J.

Definition 10 *Let the set of terminals in a language L be denoted by $\Sigma = \{a_1, a_2, ..., a_n\}$. Define a mapping Φ from $\Sigma^\star$ into J^n by*

$$\Phi(w) = (\sharp a_1(w), \sharp a_2(w), ..., \sharp a_n(w))$$

where $\sharp a_i(w)$ denotes the number of occurrences of a_i in a word w from the language L. For any $L \subseteq \Sigma^\star$ define

$$\Phi(L) = \{\Phi(w) \mid w \in L\}$$

$\Phi(L)$ is the **commutative map of L**.

Definition 11 *Two languages L_1 and L_2 defined on the same alphabet are* **letter-equivalent** *if $\Phi(L_1) = \Phi(L_2)$.*

Definition 12 *A subset Q of J^n is said to be* **linear** *if there exists members $\alpha, \beta_1, \beta_2, ..., \beta_m$ of J^n, such that*

$$Q = \{x \mid x = \alpha + n_1\beta_1 + n_2\beta_2 + ... + n_m\beta_m, \quad n_i \in J\}$$

Q is said to be **semi-linear** *if Q is a union of a finite number of linear sets.*

To be able to interpret Parikh's theorem it is necessary to have the theorem below following Harrison [27].

Theorem 3 *Let there be given an alphabet Σ. A set $L \subseteq \Sigma^\star$ has a semi-linear image if and only if L is letter-equivalent to a regular set.*

Showing that a language has a regular language representation can be done by showing that it is a semi-linear set. This is exactly the way Parikh have addressed the problem [35].

Parikh's Theorem *Let G be a context-free grammar generating the language L. Let $\Phi(L)$ be the commutative map of L. Then $\Phi(L)$ is a semi-linear subset Q of J^n.*

It is possible to interpret the result of Parikh's Theorem in the following way: Two strings are letter-equivalent, if they have the same number of occurrences of each symbol. Parikh's Theorem states that every context-free language is letter-equivalent to a regular set. As every regular set can be generated by a regular grammar a regular grammar exists which can generate strings with the same occurrence of symbols as strings in any context-free language.

3.3.1.1 Why Parikh's theorem works.

To show that the commutative map of a context-free language is a semi-linear set can be done by making use of the fact that the union of two semi-linear sets is a semi-linear set. This allows for a partitioning of the set of all strings in the context-free language into subsets.

First the set of nonterminals, N is partitioned into all possible subsets of N with the additional constrain that every subset must contain the starting symbol S. Each subset is called N^i. L is partitioned into a set of equivalence classes, L_i each class defined using the relation that two elements are in the same class if and only if, both are using the same set of nonterminals, N^i, in the derivation process. The language L becomes the union of all L_i, $L = \bigcup L_i$.

To show that L is a semi-linear set it requires that L_i is a union of linear sets, and in order to be linear a set has to consist of a finite set of terms. How is it possible in a finite number of terms to represent a possible infinite set of strings? This is not possible in general, but the number of occurrences of symbols within the set of strings can be represented. This can be done by breaking all strings down into a finite set of basic building blocks and then counting the number of times they are used.

The breakdown in a finite set of building blocks is based on Ogden's lemma or the pumping lemma for context-free languages, which says that any context-free language containing strings of length above a certain language dependent constant n, must contain loops in the derivation process, i.e., the derivation trees of such strings will contain repetitions of subtrees along at least one path through the tree. For each L_i two sets of basic building blocks R_{A_i} and T_S are created.

R_{A_i}: Subtrees which for all $A_i \in N^i$ derive a sentential form $A_i \overset{\star}{\Rightarrow} w_1 A_i w_2$, using only nonterminals from N_i. The substrings w_1 and w_2 contain only terminals. Such subtrees are capable of repeating the roots of the tree. It is argued that a upper limit for the number of specific nonterminals along any path of the derivation tree exists. Above the limit only repetitions will occur. The upper limit ensures that the size of the tree is bounded and that the total number of such trees therefore is limited.

T_S: Derivation-trees describing derivations of the form $S \overset{\star}{\Rightarrow} w$, where w contain no nonterminals and the derivations use only nonterminals from N^i. Using the same arguments as above it is argued that the number of such trees is finite.

It is now argued that based on combinations of the two sets any string in L_i can be constructed. The idea is that any nonterminal in a tree in T_S can be replaced by one of the trees in R_{A_i} and each time a new string z containing the symbols from w_1, w_2, and w is obtained.

For each string w in T_S we can represent the commutative image of all strings, Q_w, derivable by extensions of the derivation tree of w in the following way;

$$\Phi(Q_w) = \Phi(w) + \sum n_{i1}^1 V_{i1}^{A_1} + \sum n_{i2}^2 V_{i2}^{A_2} + \cdots + \sum n_{in}^n V_{in}^{A_n}$$

where $V_k^{A_i}$ is the commutative image of the kth derivation tree associated with $A_i \overset{*}{\Rightarrow} w_1 A_i w_2$, $n_i^j \in \{1, 2, 3, 4, \cdots\}$. $\Phi(Q_w)$ is clearly linear and since the set of strings, w_j in T_S is finite then

$$\Phi(L_i) = \bigcup \Phi(Q_{w_j})$$

When $\Phi(L_i)$ is a union of linear sets it is per definition a semi-linear set, and it is now the case that

$$\Phi(L) = \bigcup \Phi(L_i)$$

which shows that the commutative image of a context-free language is a semi-linear set.

The general idea behind Harrison's theorem showing that a semi-linear set can be represented by a regular language/grammar is that a regular grammar which per definition must have a finite set of terminals can be constructed so there is a one-to-one correspondence between each terminal symbol a_i and each of the terms $\Phi(w)$ and $V_i^{A_i}$ in $\Phi(L)$ and through self-embedding productions of the form $A \Rightarrow a_i A$ any number, n_i^j, of occurrences of a_i can be generated.

For details about the formal proofs the reader is directed to Parikh [35] and Harrison [27].

3.3.2 Pilling's Method

Parikh's Theorem has been considered one of the most fundamental theorems of context-free languages [11, 35] due to its theoretical importance. Parikh was, however, not able to point out an efficient algorithm to obtain the regular language representation. Such an algorithm was provided several years later by Pilling [38]. He provided a new interpretation of Parikh's Theorem, as a theorem of solution to regular expression equations, and described an efficient method for obtaining a regular language representation of a context-free language.

Regular expressions were first defined by Kleene. The following definition is due to Aho and Ullman [1]:

Definition 13 Regular expressions *over Σ and the regular sets which they denote are defined recursively as follows:*

1. *$\emptyset$ is a regular expression denoting the regular set $\{ \emptyset \}$.*
2. *λ is a regular expression denoting the regular set $\{ \lambda \}$.*

3. *a in Σ is a regular expression denoting the regular set $\{ a \}$.*
4. *If p and q are regular expressions denoting the regular sets P and Q, respectively then*
    - *$(p+q)$ is a regular expression denoting $P \cup Q$.*
    - *(pq) is a regular expression denotingPQ.*
    - *$(p)^\star$ is a regular expression denoting $P^\star$.*
5. *Nothing else is a regular expression.*

Based on regular expressions it is possible to define regular functions, and regular expression equations.

Definition 14 *A* **regular function** *is any function over a set of regular expressions based on the three operations, previously defined; (p+q), (pq) and $(p)^\star$, where p and q are regular expressions.*

Definition 15 *Let $\Delta = \{x_1, x_2, \cdots, x_n\}$ be variables representing sets.*
Let $\Psi = \{a_1, a_2, \cdots, a_n\}$ be regular expressions.
A regular expression equation *is an equation over the set of variables Δ so that the coefficients are in Ψ.*

An algorithm for converting from a context-free grammar to a regular expression equation is given in Algorithm 2.

Before Pilling's Method was published in 1973, methods existed for solving regular expression equations created from a regular grammar. An extension to context-free languages would require that two problems were solved; 1) The ordering of terminals and nonterminals can be arbitrary and 2) multiple occurrences of any nonterminal can take place, including the one that the equation is to be solved for.

The first problem is related to the fact that the symbols in an equation must be ordered in a special way for a solution to be obtainable. The problem can not be solved in general, but in the case of Parikh's Theorem where commutativity is assumed, Pilling shows that the commutative map of a language generated by a grammar G is the same as for the language generated by a commutative set of symbols, i.e., nonterminals and terminals so that the ordering of symbols is not fixed. They can then be moved freely around to ensure the special required ordering.

To address the second problem it is advantageous to view how Pilling describes an equation. Any equation can be on the form:

$$A_0 = f_0(N, \Sigma)$$

where f_0 is a regular function on the nonterminals and the terminals, that is, any number of terminals or nonterminals can occur. He then separates the terms of the equation into two groups, those involving A_0 and those not involving A_0.

$$A_0 = E(N \setminus \{A_0\}, \Sigma) + F(N, \Sigma).A_0$$

where E is a function describing all terms in which A_0 does not take part. F is a function based on all terms in which A_0 occurs at least once. $F(N, \Sigma).A_0$ means that $F(N, \Sigma)$ is concatenated with A_0, where A_0 is moved to the right end of the substring by use of commutativity. According to Pilling the equation has the solution:

$$A_0 = [F(N, \Sigma) \mid_{A_0 = E(N \setminus \{A_0\}, \Sigma)}]^{\star} E(N \setminus \{A_0\}, \Sigma)$$

In short form it is written as:

$$A_0 = [F(E)]^{\star} E$$

From the solution it is evident that Pilling solves the second problem by considering nonterminals other than the one for which the equation is to be solved, A_0, as a constant, and all occurrences of A_0 are substituted with the set of terms not directly related to A_0.

An algorithm for solving a single equation is given in Algorithm 3.

In the following examples the simplicity and efficiency of Pilling's Method are illustrated.

Example 1.

Equation: $A = abA + c$
Solution: $A = (ab)^{\star}c$
Where c represents E and ab represents F.

Example 2.

Equation: $A = abAAA + c$
Solution: $A = (abcc)^{\star}c$
Where c represents E and $abAA$ represents F.

Example 3.

Equation: $A = aABAc + e + d$
As the alphabet is commutative the equation can be rewritten.
Equation: $A = acBAA + e + d$
Solution: $A = (acB(e + d))^{\star}(e + d)$
Where $(e + d)$ represents E and $acBA$ represents F.

In most cases where a grammar is used to derive a regular expression equation more than a single equation exist. There will, in general, be one equation for each nonterminal in the grammar. It is therefore of interest to see how Pilling's Method solves a set of equations. The basic idea is that a set of equations can be solved one equation at a time, treating the other nonterminals as constants. The system of equations can be solved using the idea of Gauss's elimination known from traditional systems of equations. An algorithm is given in Algorithm 4.

Example 4:

Let there be given a grammar represented as the following set of equations:

$$\begin{aligned}
S &= aSSA + b \\
A &= cAB + d \\
B &= eSB + f
\end{aligned}$$

Following Algorithm 4 the equation for S is solved first and the result is substituted in the equations for A and B.

$$\begin{aligned}
S &= (aSA)S + b = (abA)^\star b \\
A &= cAB + d \\
B &= e(abA)^\star bB + f
\end{aligned}$$

A solution for A is found and inserted in B and B is thereby solved.

$$\begin{aligned}
S &= (abA)^\star b \\
A &= (cB)A + d = (cB)^\star d \\
B &= [e(ab(cB)^\star d)^\star b]B + f = [e(ab(cf)^\star d)^\star b]^\star f
\end{aligned}$$

The solution for B is now inserted in A.

$$\begin{aligned}
S &= (abA)^\star b \\
A &= (c[e(ab(cf)^\star d)^\star b]^\star f)^\star d \\
B &= [e(ab(cf)^\star d)^\star b]^\star f
\end{aligned}$$

The last step is to insert the result for A in S.

$$\begin{aligned}
S &= (ab(c[e(ab(cf)^\star d)^\star b]^\star f)^\star d)^\star b \\
A &= (b[e(ab(cf)^\star d)^\star b]^\star f)^\star d \\
B &= [e(ab(cf)^\star d)^\star b]^\star f
\end{aligned}$$

Assuming that S is the starting symbol in the grammar, the solution for equation S will be a regular language representation of the language from which the equations are produced.

3.3.3 The requirements

As the basic idea of the regular language representation is to represent a stochastic language it is necessary to investigate, if the strings in the regular language can still be given the correct probabilistic interpretation. A string in the regular language representation can represent several strings in the original context-free language. In order to have a correct probabilistic interpretation the string must have a probability equal to the probability for all the strings represented by the string. It is here important to remember that as stated in Definition 6 in Chapter 2, the probability of a string is the sum of probabilities for all different derivations of a string.

The problem to address therefore becomes one of determining, given a commutative image of a string, how many different derivations can be generated by the string and to ensure that they all are accounted for. There are two different cases:

1. The grammar can be ambiguous so that different derivations lead to the same non-commutative string.
2. The grammar is un-ambiguous, but properties of the derivation process can create different trees having identical commutative maps.

3.3.3.1 Case 1.

This case does not constitute a problem as it can be proven that a derivation language can not be ambiguous as stated in Theorem 4.

Theorem 4 *Any derivation sequence language DL(L) of a context-free language L is unambiguous.*
There is only one string in DL(L) for each production sequence in L, but if L is ambiguous each of the production sequences will have a string in DL(L).

Proof:
The idea of the proof is to show that a string which *is* ambiguous can not be generated by a derivation language.
Let the string $w = a_1 a_2 \cdots a_k$ be ambiguous, meaning that at least two different derivation trees exist for w. For the trees to be different the two trees must have a different breakdown in subtrees. Let the two subtrees, which are different but are creating the same string, have a common root denoted A_m. If two trees having A_m as the root are to have different configurations, A_m must have multiple ways to be rewritten, e.g.,

$$A_m \to a_i A_i \cdots A_k$$

$$A_m \to a_j A_j \cdots A_n$$

where the ordered sequence $(A_i \cdots A_k)$ is different from $(A_j \cdots A_n)$. Starting with A_m and using different rewritings to get to the same frontier of the derivation tree requires that $a_i = a_j$. As each production has one unique label being the terminal symbol, such an requirement can not be fulfilled by a derivation grammar.

End of proof.

3.3.3.2 Case 2.

For two derivation trees to be different and still have the same commutative image the situation depicted in Figure 3.1 must occur. Somewhere in the derivation tree two identical nonterminals A_1 and A_2 having subtrees T_1 and T_2 exist, giving rise to substring w_1 and w_2. If $w_1 = w_2$ it only counts as one tree. If, however, $w_1 \neq w_2$ it is possible that T_1 and T_2 can be interchanged, giving rise to two different trees having the same commutative image. In general two ways exist for introducing such multiple occurrences of identical nonterminals, 1) through a derivation like $S \overset{\star}{\Rightarrow} z_1 A z_2 A z_3$ or 2) through a derivation like $A \overset{\star}{\Rightarrow} z_1 A z_2 A z_3$. It is assumed that 1) does not have situation 2) as a subset of its derivations.

Situation 2) is recognised as the situation characterizing the expansive grammars, and within the scope of this method this situation is of no concern.

For situation 1) it can be shown that it constitute no problem in relation to Pilling's Method, as stated in Theorem 5.

Theorem 5 *All derivation trees for a non-expansive context-free derivation language leading to strings having identical commutative maps will be accounted for when Pilling's Method is used to obtain a regular language representation.*

Proof:
Consider the situation where the multiple occurrences of a nonterminal is due to a derivation like $S \overset{\star}{\Rightarrow} z_1 A z_2 A z_3$. For multiple occurrences to create a special

problem A must be able to be rewritten in different ways. Let the different ways be represented by the following equation.

$$A = aX + bY + \cdots + cZ + dA + eA$$

where $X, Y, \cdots, Z$ is in $(N \setminus \{A\})^\star$. When such an equation is solved using Pilling's Method the results become:

$$A = (d + e)^\star(aX + bY + \cdots + cZ)$$

From the definition of the union-operator $(+)$ all possible combinations of the terms will be accounted for, i.e., both (aX, bY) and (bY, aX). Each of the terms in a regular expression can be thought of as a sub-derivation tree, which proves that all configurations of subtree are accounted for using Pilling's Method.

End of proof.

It can be concluded that all requirements for a probabilistic interpretation of the strings in a regular language representation of a non-expansive derivation language are fulfilled. This is, however, not the case for expansive grammars which will be discussed i Section 3.6.

3.4 Step three

The last step is to compute the characteristics of interest for the stochastic language, based on the solution of the set of regular expression equations. In order to do so it is necessary to 1) change representation for the solution of the equations and 2) to reintroduce the stochastic interpretation of the production labels.

The change of representation is necessary as the form of the solution is in set-theoretic terms which is not easy to work with. An obvious choice for a representation is a generating function, which is known to be a powerful tool and a natural way of expressing language properties.

Definition 16 *Let a_0, a_1, a_2........ be a sequence of real numbers. The function*

$$F(x) = a_0 + a_1x + a_2x^2 + \cdots = \sum_{i=0}^{\infty} a_ix^i$$

is called the **generating function** *for the given sequence.*

The basic idea is that there will be one generating function for each of the properties that will be investigated. The dummy-variable x will be associated with a random variable RV, and the coefficient to x^n will be the probability of n occurrences of the random variable.

The change in notation is possible, since the following rules for converting between the set-domain and generating functions exist [28].

In a regular expression like $(p_1p_2 + p_4)^\star p_4$, p_i will denote the set $\{p_i\}$. Therefore the generating function for $\{p_i\}$ can be found to be p_i, so the individual

	Set	Gen. Funct.
Union	C = A + B	$F_C = F_A + F_B$
Cartesian prod.	$C = A \times B$	$F_C = F_A F_B$
Sequence of	$C = (A)^\star$	$F_C = 1/(1 - F_A)$

Table 3.1: Relations between set operators and generating functions.

symbols are generating functions for the set which they represent. The union of two sets becomes the sum of the generating functions for each of the two sets. The Cartesian product is the operation, which in regular expression notation is concatenation so the concatenation of two regular expressions can be represented by the product of the two generating functions. Finally the very important star operator $(A)^\star$, can be represented as $1/(1 - F_A)$, which when $\{A\}$ is represented by its generating function A, can be written as

$$1 + A + A^2 + A^3 + \cdots = \sum A^n$$

This shows that any star operation in a solution to a regular expression equation can be represented as a sum of infinitely many terms.

Having established a connection between the two domains, it is now necessary to look at how the new representation is constructed correctly to represent a specific property of the language and to look at how the statistical information is brought into the representation. The basic idea for each of the symbols in the solution is to make an inquiry into each production of the stochastic grammar concerning the number of occurrences that the production gives rise to regarding the random variable of interest. Let, as an example, the random variable denote the number of occurrences of terminals in a string. Let there be given two stochastic productions $p_1 : S \to aaS$, $p_2 : S \to b$, then assign a x^2 to the symbol p_1 and a x^1 to p_2. The power of x represents the number of occurrences which each production contributes to the random variable; i.e., production p_1 contributes two terminals and p_2 one terminal.

If this procedure is done for every symbol in the generating function representation of the solution to the system of equations, the result will be a generating function capable of generating a sequence where the coefficients to the dummy-variable x^n denote the probability mass associated with the set of strings having n terminals.

Algorithm 5 describes the conversion from a regular expression to a generating function.

When the generating function representation is obtained, information about the mean and the variance for the random variable can be computed as:

$$E(RV) = GF'(x)_{|x=1} = GF'(1)$$

$$Var(RV) = GF''(1) + GF'(1) - GF'(1)^2$$

An example of the use of Algorithm 5 is given in the next section.

3.5 Computing characteristics

It is very important for understanding the potential and the limitations of the method to realize that it is an enumerative technique based on a controlled enumeration of strings in a derivation language, representing strings in the stochastic language of interest. First, a simple example shows how characteristics for a grammar can be computed, and then some more advanced examples will be outlined in order to discuss the limitations of the method due to the ennumerative approach.

3.5.1 A simple example

Let there be given a stochastic context-free grammar $G = (\{S, A\}, \{a, b, c, d\}, P, S)$ where P has productions:

$$\begin{array}{ll} p_1 : S \to aSA & p_2 : S \to cd \\ p_3 : A \to b & \end{array}$$

Let $p_1 = 0.7$, $p_2 = 0.3$ and $p_3 = 1$.

Using step one G is changed into a non-stochastic derivation grammar $DG = (\{S, A\}, \{p_1, p_2, p_3\}, P', S)$, where P' has productions

$$\begin{array}{ll} S \to p_1SA & S \to p_2 \\ A \to p_3 & \end{array}$$

Using step two the grammar is changed into a set of regular equations. There is one equation for each nonterminal.

$$\begin{array}{l} S = p_1SA + p_2 \\ A = p_3 \end{array}$$

Substituting the value for A into the equation for S and allowing S to commute the equation becomes:

$$S = p_1p_3S + p_2$$

Using Pilling's Method the solution becomes:

$$S = (p_1p_3)^\star p_2$$

In step three the solution from step two is represented as a generating function. There is one generating function for each of the random variables of interest. In this example the generating function for the length of strings is used. The variable x in the function denotes the number of occurrences of terminals introduced when a production is used.

$$\begin{aligned} Gen.funct.(string\ \ length) &= \sum_{i=0}^{\infty}(p_1xp_3x)^i p_2x^2 \\ &= p_2x^2 + p_1p_2p_3x^4 + p_1^2p_2p_3^2x^6 \ldots \end{aligned}$$

The x's associated with p_1 and p_3 are raised to power one, since the productions associated with these symbols only contribute one terminal to the length of the string. The x associated with p_2 is raised to the power of two, since the production adds two terminals to the string every time it is used.

When the symbols are now given their numerical value the information directly available from the generating function is:

A string of length 2 exists with probability $p_2 = 0.3$
A string of length 4 exists with probability $p_1p_2p_3 = 0.21$
A string of length 6 exists with probability $p_1^2p_2p_3^2 = 0.063$
$\vdots$

Based on the generating function information about the mean and variance can also be computed , E(string length) = 6.66 and Var(string length) = 31.11.

3.5.2 General considerations

For enumerative techniques it is often the case that the total number of enumerations can be very large and for all enumerative techniques there is a problem of determining how many enumerations/iterations are required to get the result with a given accuracy. This is strongly related to the structure of the grammar and therefore related to the structure of the solution obtained by Pilling's Method. Some typical representative solutions for the generating functions can be listed,

1) $\sum_{i=0}^{\infty}(p_1p_2x)^ip_3$

2) $\sum_{i=0}^{\infty}(p_1+p_2x)^ip_3$

3) $\sum_{i_1=0}^{\infty}(p_1p_3)^{i_1}\sum_{i_2=0}^{\infty}(p_4p_2x)^{i_2}p_3$

4) $\sum_{i_1=0}^{\infty}(p_1p_2x)^{i_1}\sum_{i_2=0}^{\infty}(p_4p_2x)^{i_2}p_3$

5) $\sum_{i_1=0}^{\infty}((p_1p_2x)(\sum_{i_2=0}^{\infty}(p_4p_3)^{i_2}))^{i_1}p_3$

6) $\sum_{i_1=0}^{\infty}((p_1p_2x)(\sum_{i_2=0}^{\infty}(p_4p_2x)^{i_2}))^{i_1}p_3$

Case 1: This case contains no enumerative problems as each new step in the summation gives a new value of some instance of the random variable x. The enumerations must continue until the instance of interest is reached.

Case 2: This simple case touches upon a fundamental enumerative problem. Those strings which should be counted are spread out over the entire set of strings. Let us consider an example.

Let there be given a stochastic context-free grammar $G = (\{S\}, \{a, b, c\}, P, S)$ where P has productions:

$$p_1 : S \to aS \qquad p_2 : S \to bS$$
$$p_3 : S \to c$$

Let $p_1 = 0.3$, $p_2 = 0.2$ and $p_3 = 0.5$. If we obtain the solution and express it in terms of a generating function for the number of occurrences of e.g., terminal b we get the following,

$$\text{Gen.funct.(number of } b\text{'s)} = \textstyle\sum_{i=0}^{\infty}(p_1 + p_2x)^i p_3$$

Let us consider the situation in which we want to compute the probability of strings containing one b. Every time we increase the enumeration index i we derive new strings having one occurrence of a b. To get the correct probability each new string should be added to the set of strings on which the accumulated probability of strings having one b is computed. Looking at the added terms for different values of the enumeration index i we get,

$i = 0$:	p_3
$i = 1$:	$p_1p_3 + p_2p_3x$
$i = 2$:	$p_1^2p_3 + 2p_1p_2p_3x + p_2^2p_3x^2$
$i = 3$:	$p_1^3p_3 + 3p_1^2p_2p_3x + 3p_1p_2^2p_3x^2 + p_2^3p_3x^3$
$i = 4$:	$\cdots 4p_1^3p_2p_3x \cdots$
$i = 5$:	$\cdots 5p_1^4p_2p_3x \cdots$

Note that as p_1, p_2, p_3 are probabilities they represent values less than 1. Therefore the resulting new additional term decreases rapidly in numerical value as the index i increases. Any level of accuracy can be achieved by including more and more terms. In practice a threshold on the increment in probability mass achieved by going from $i = k$ to $i = k + 1$ would be applied. In the example setting a threshold $\epsilon = 0.0001$, 8 iterations is required. Going from $i = 8$ to $i = 9$ gives an increase of 0.000059. With 8 iteration we get,

$$\text{P(strings with one } b) = 0.203993$$

As can be seen from the listing of the terms the relevant terms in this case can be expressed as

$$\text{P(string with one } b) = \textstyle\sum_{n=1}^{\infty} np_1^{n-1}p_2p_3$$

If Maple is used to evaluate the sum from $n = 1$ to infinity the correct result can be found to be 0.204081. This result clearly indicates that good approximations can be achieved by using a limited number of iterations in the theoretically infinite enumeration process.

Case 3 and 5: In these cases the strings are a result of two independent summations. As the random variable of interest x is related to only one of the summations, it is possible to evaluate explicitly the probability mass generated by the other summations and use that result as a constant in the computations of the random variable. Otherwise these cases follow case 1.

Case 4 and 6: In these cases the random variable of interest is associated with both summations. Following the line of thoughts of case 2 an approximation will be the only feasible solution.

General considerations: It is considered that most real situations will result in configurations of the regular language representation for which an approximation of the values for the random variables is the only realistic result. As the approximation can be obtained with any accuracy required this is not considered a significant drawback of the method.

More information on the applicability of the method is given in Chapter 5.

3.6 Expansive grammars and Pilling's Method

The reason why the method presented in this chapter is restricted to the non-expansive grammars is that expansiveness constitute a very special situation, and that expansive grammars create situations in which a probabilistic interpretation of the string in the regular language representation is not possible. The problem is that since a nonterminal A_i can introduce at least two occurrences of itself during one or more rewritings, a mechanism exists by which derivation trees for strings of even moderate size can have occurrences of the same nonterminal A_i in many different paths in the tree. For each of those nonterminals subtrees can develop. Since they have a common root A_i, they can be interchanged and still maintain the same commutative image. As will be discussed in the next section, there is a fundamental conflict between the fact that the regular language representation by Pilling is based on strings where the ordering of subtrees within the derivation tree is disregarded, and the fact that the ordering of trees must be known to compute the correct probability of a string.

To give a few examples of the problem Figure 3.2 shows some general configurations having multiple occurrences, and Figure 3.3 shows one of the situations in detail where it can be seen how different configurations give the same commutative map.

3.6.1 Documentation for the problem

It will now be proven that strings in a regular language representation of expansive context-free languages obtained by using Pilling's Method can not directly be given a probabilistic interpretation. First it is shown in Theorem 6 and 7 that for no expansive grammar the strings can be given a correct probabilistic interpretation. Theorem 6 states that an equation, which in its right hand side has multiple occurrences of the variable, for which the equation is to be solved, always occurs when solving the set of equations for an expansive grammar. Theorem 7 shows that the count of derivation trees is incorrect, when Pilling's Method is used to solve an equation with such multiple occurrences. Secondly it is shown in Theorem 8 that such problems occur only for expansive grammars. The knowledge is summed up in Theorem 9.

Theorem 6 *Let there be given a context-free grammar which is commutative and expansive for symbol A. Let the grammar be represented as a set of regular expression equations. Then there will be an equation, either directly or by* *reduction of the set of equations, having the form:*

$$A = F(X, Y, \ldots, Z, A).AA + E(X, Y, \ldots, Z, A)$$

F and E are regular functions. X, Y,...,Z are nonterminals or terminals. A is a nonterminal.

Proof: There are 2 cases in which the equation can appear:

- Directly. This is the trivial case where the grammar contains a production of the form:

  $$A \rightarrow \alpha AA$$

- Indirectly. This is the case where a number of productions has to be used for the grammar to be expansive.

  In such a case the equation for A, before any reduction of the set of equations could be specified as:

  $$A = F(X, Y, \ldots, Z, A) + E(X, Y, \ldots, Z)$$

  Since the grammar *is* expansive for A, some of $(Z, Y, \ldots, Z)$ must, directly or through a chain of dependencies, be a function of A.

  Pilling [38] proved that a set of regular equations can be solved one equation at a time, treating the other variables as constants.

  The multiple occurrences of A are a result of a sequence of rewritings of nonterminals:

  $$A \Rightarrow uX_1vY_1w \Rightarrow uX_2vY_2w \Rightarrow \cdots \Rightarrow uX_nvY_nw \Rightarrow uAvAw$$

  Since X_n, Y_n can be rewritten as A, A is a part of the equation for X_n and Y_n. While solving those 2 equations A will be kept constant and therefore becomes a part of the solution for X_n, Y_n.

  Since X_{n-1}, Y_{n-1} can be rewritten as X_n and Y_n respectively, X_n, Y_n is a part of the equation for X_{n-1}, Y_{n-1}. Since X_n, Y_n is kept constant, and A is a part of X_n, Y_n, it also becomes a part of the solution for X_{n-1}, Y_{n-1}.

  In a similar way, it is guaranteed that the multiple occurrences of A become a part of the equation for A. Since the language is commutative, the symbols can be rearranged giving the equation:

  $$A = F(X, Y, \ldots, Z, A).AA + E(X, Y, \ldots, Z, A)$$

End of proof.

Theorem 7 *Given a regular equation of the form:*

$$A = F(X, Y, \ldots, Z, A).AA + E(X, Y, \ldots, Z, A)$$

where F and E are regular functions. (X, Y,...,Z) are nonterminals or terminals of the underlying grammar.
If the equation is solved using Pilling's Method, not all different derivation trees leading to the same commutative image are detected.

ALGORITHM 2. Conversion to a Regular Expression Equation.

Input:

Output:

Method:

A context-free grammar $G = (N, \Sigma, P, S)$, specified as
$N = \{A_1, A_2, \cdots, A_n\}$, $\Sigma = \{a_1, a_2, \cdots, a_m\}$, $S = \{A_1\}$
The production must be of the following form:
$A_i \rightarrow \beta$

A set of regular expression equations. The number of equations is equal to the number of nonterminals in N.

```
1)    / * Computes the equivalence classes */
      For all productions $P$ do
          if $P = A_k \rightarrow \beta$ then
              $C_{A_k} = C_{A_k} \cup \{A_k \rightarrow \beta\}$
          End
      End
2)
      / * Create the regular exp. equations */
      For each $C_{A_i}$ do
          For each element in $C_{A_i}$ do
              $A_i = A_i + \beta$
          End
      End
```

ALGORITHM 3. Solving a single regular expression equation.

Input:

Output:

Method:

A single equation on the form
$A = \beta_1 + \beta_2 + \cdots + \beta_k$, where $\beta_k \in (N \cup \Sigma)^\star$.

$A = f(\Sigma)$, where $f(\Sigma)$ is a regular function on the set of terminals.

1) Bring
$$A = \beta_1 + \beta_2 + \cdots + \beta_k$$
on the form
$$A = \alpha A + \beta,$$
where $\alpha = f(N, \Sigma)$ and $\beta = f(N \setminus \{A\}, \Sigma)$

2) Change the equation into the form
$$A = \alpha^\star \beta$$

3) Substitute A in α with β.

4) Output the resulting equation.

ALGORITHM 4. Solving a system of regular expression equations.

Input:

Output:

Method:

A set of n equations each on the form
$A_i = \beta_1 + \beta_2 + \cdots + \beta_m$, where $\beta_k \in (N \cup \Sigma)^\star$.

A set on n solutions each on the form
$A_i = f(\Sigma)$, where $f(\Sigma)$ is a regular function on the set of terminals.

```
1)  Set $i = 1$
2)  **If** $i = n$ go to step 3
    **Else**
        Apply Algorithm 3 to the $i$th equation and
        Substitute the result into the $(i+1)$th .. $n$th equation.
        Increase $i$ and return to step 2.
    **End**
3)
    $i$ is now equal to $n$, move to step 4.
4)
    The equation for $A_i$ only contains terminals now.
    Output the equation for $A_i$ and substitute the result for $A_i$
    in the remaining equations.
5)
    **If** $i = 1$ Stop
    **Else**
        Decrease $i$ by 1 and return to step 4.
    **End**
```

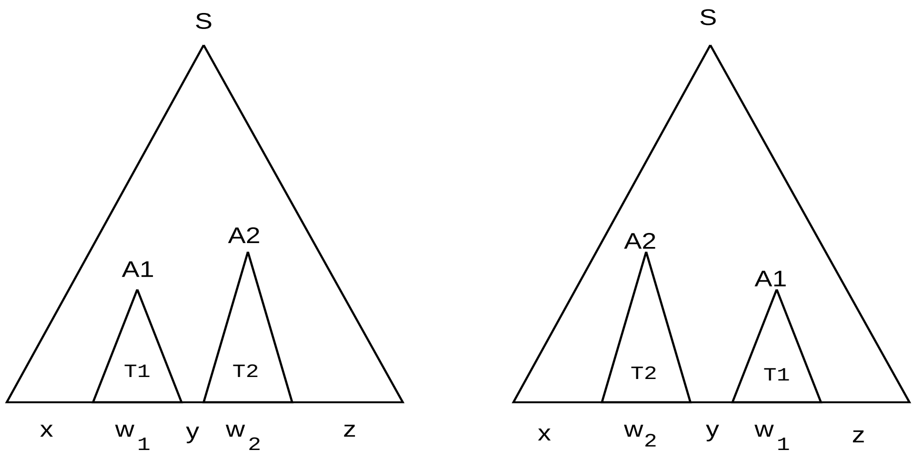


Figure 3.1: A situation in which two different derivation trees have identical commutative maps.

ALGORITHM 5. From a regular expression to a gen. function.

Input:

Output:

Method:

A stochastic context-free grammar $G = (N, \Sigma, P, S)$.
E, a solution to the system of regular expression equations for the grammar G.
RV, a random variable of interest in relation to G or $L(G)$.

A generating function GF for the random variable RV.

```
1)  /* Convert from $E$ to $GF$ */
    Change terms like $(p_i + p_j)$ in $E$ to $F(p_i) + F(p_j)$ in $GF$.
    Change terms like $(p_i p_j)$ in $E$ to $F(p_i)F(p_j)$ in $GF$
    Change $(p_i)^\star$ from $E$ into $1/(1 - F(p_i))$ in $GF$
    Set F$(p_i) = p_i$

2)  /* Insert the stochastic information */
    **For** each production in $P$ **do**
        Set $\xi$ equal to the number of occurrences of the random
        variable $RV$ associated with the production.
        Substitute $p_i$ with $p_i x^\xi$ in $GF$
    **End**
```

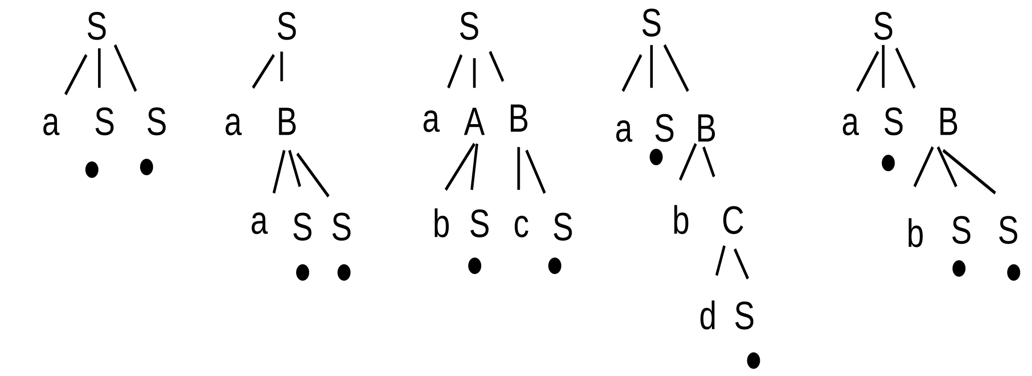


Figure 3.2: Illustrations of 5 situations which cause a grammar to be expansive. The dots indicate nonterminals from which subtrees can be developed.

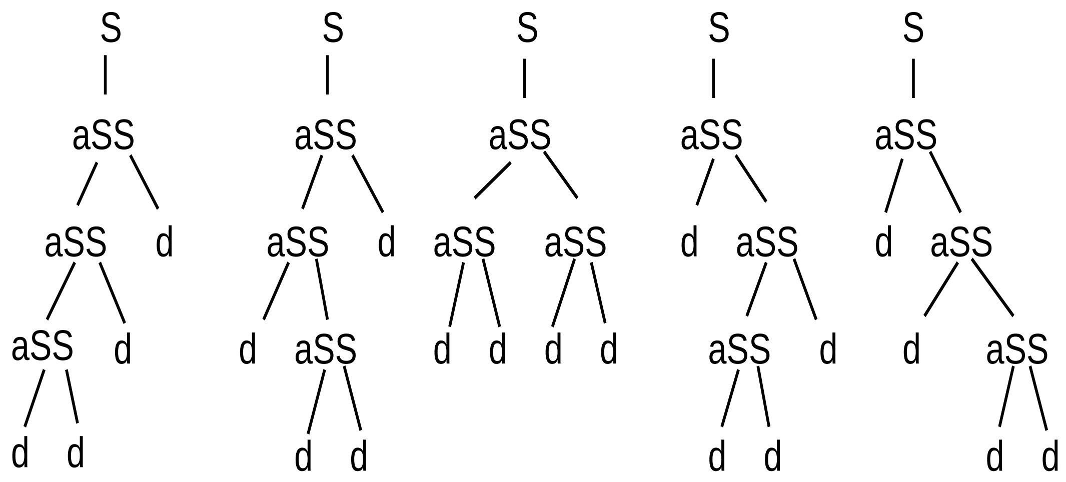


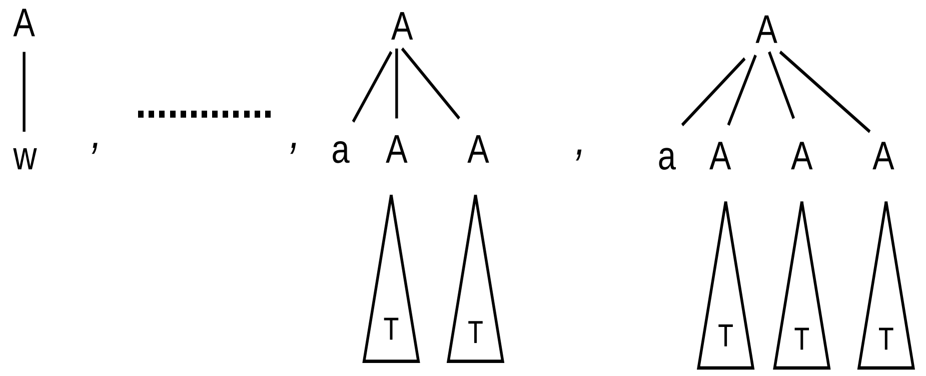
Figure 3.3: Five derivation trees derived from a grammar with productions $S \to aSS, S \to d$. All trees have a^3d^4 as its commutative image.

Proof. The equation is changed to fit the description for Pilling's Method.

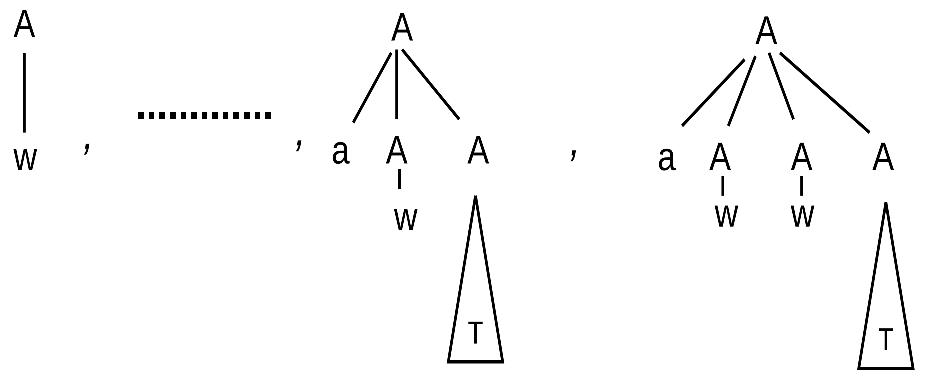
$$A = F(X, Y, \ldots, Z, A)A + E(X, Y, \ldots, Z,)$$

where $F(X, Y, \ldots, Z, A)$ now represent at least one occurrence of A. Using Pilling's Method will cause a substitution of all the occurrences of A in $F(X, Y, \ldots, Z, A)$ with $E(X, Y, \ldots, Z)$.

The individual terms in the equation can be viewed as derivation trees. The equations can be represented by sets of trees like:



The substitution in Pilling's Method will cause the trees to be changed into:



The important thing here is that the number of different ways in which subtrees can be arranged to form a string is limited, since one of the terminals from which it should develop is changed into a leaf. From the set of reduced trees it is not possible to account for all different trees leading to the same commutative image.

End of proof.

Theorem 8 *Let G' = (N,Σ',P',S) be a derivation grammar for a commutative context-free language. Let the grammar be represented as a set of equations. Use Pilling's Method to solve the set of equations. Only if the grammar is expansive, some of the derivation trees giving the same commutative image will not be accounted for.*

Proof: The problem arises because a commutative string can have different derivation trees, if the order of symbols is changed. To have different derivation trees, it is necessary for a string to have multiple occurrences of identical nonterminals among the leaves of the derivation tree at any given configuration of the tree. Such situations can occur in 2 cases:

1. Multiple occurrences without the nonterminals having a common root with the same label as the nonterminals.
2. Multiple occurrences with a nonterminal of the same kind, being a common root.

Case 1: This is the situation described in Section 3.3.3, where Theorem 5 show that it does not constitute a problem.
Case 2: The situation in this case is exactly a situation like the one described in the definition of expansive grammars. From Theorem 6 and 7 it is known that a solution using Pilling's Method will give an incorrect count of different derivation trees.
End of proof.

Theorem 9 *Let the derivation sequence language for a commutative stochastic context-free language be represented as a set of regular equations. Use Pilling's Method to solve the set of equations. If and only if the grammar is expansive the set of strings can not be given a probabilistic interpretation*

Proof: The "if" part is proven by use of Theorem 6 and 7. Since some derivation trees will not be accounted for, the correct probability can not be derived. The "only" part is proven by use of Theorem 8, which states that there are two cases in which a string in a derivation language can have different derivations. The first case can be solved correctly. The second occurs only for the expansive grammars.
End of proof.

Pilling's Method solves regular expression equations. If the equation has multiple occurrences of the variable for which it is to be solved, it fails to give the correct count of derivation trees because, all but one is substituted with a constant. This can directly be related to the development of derivation trees; whenever multiple occurrences of the nonterminal take place, all but one is forced to be rewritten as a terminal. Thereby the tree is only allowed to develop from one nonterminal. This clearly limits the choices of combinations of productions, and the count of different derivation trees is therefore corrupted.

It is important to note that Pilling's Method counts all strings having different commutative image. This is possible because any tree starting with the production, which was forced to terminate, can occur as a subtree within the subtree derived from the nonterminal which was not forced to terminate. An illustration is given in Figure 3.4.

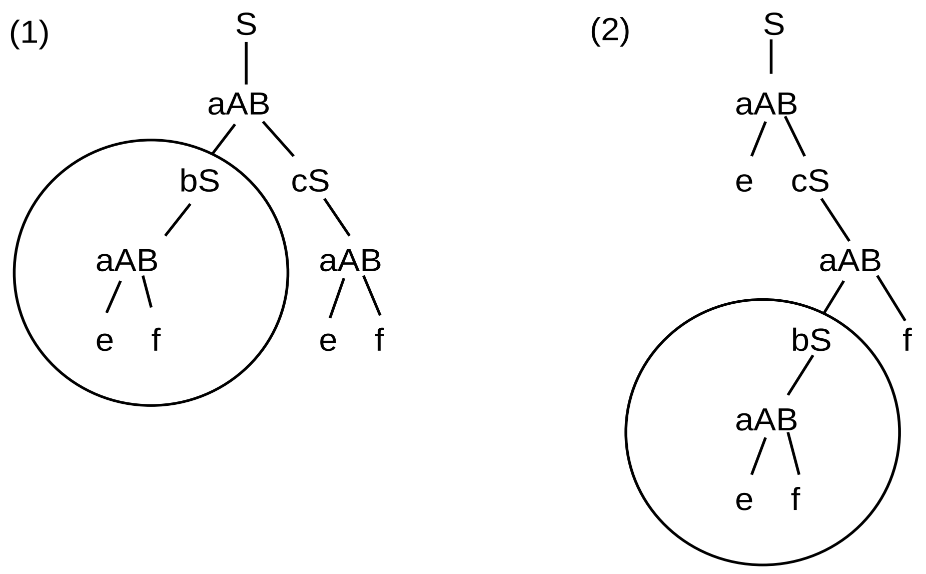


Figure 3.4: Illustration of two derivation trees having the same commutative image.

Tree no. 1 would not be among those accounted for by Pilling's Method, because for example the S introduced by production $A \rightarrow bS$ would have been forced to terminate. As shown by tree no. 2 there exists a tree which is accounted for by Pilling's Method, which has the same commutative image as tree no. 1. Note that the subtree starting with bS in tree no. 1 can be found as a subtree derivable from production $B \rightarrow cS$ in tree no. 2.

3.7 Summary

The hypothesis proposed in Section 3.1 has been verified. The three-step-procedure does provide an approach by which statistical order-independent characteristics can be computed for stochastic regular and non-expansive stochastic context-free languages.

The new method makes use of a derivation grammar in order to map from the stochastic domain into a non-stochastic (Algorithm 1). In this domain the context-free grammar is represented as a set of equations and a solution by Pilling's Method provide a regular language representation (Algorithm 2, 3 and 4). The solution is expressed in terms of a generating function which can be designed explicitly for each random variable of interest (Algorithm 5). The output of the generating function can be given a statistical interpretation.

The chapter provides new insight into how regular language representations for non-stochastic languages can be used in the stochastic domain. This involves developing a strategy for changing from the stochastic to the non-stochastic domain and back in such a way that some statistical information of the language can be maintained. Furthermore it has been shown that a regular language representation based on Pilling's Method can not handle the complexity of expansive context-free grammars in such a manner that the statistical interpretation of the obtained results is possible.

By the design of the new method it has been made possible also to use an enumerative technique for the computations of characteristics. It is based on a controlled enumeration of strings in the language instead of the random generation of strings, resulting from using the grammar directly for the enumeration. Enumerative methods have the advantage of allowing the computation of characteristics related to a subset of strings in the language.

Chapter 4

A method for expansive stochastic context-free languages

To fulfill the goal pointed out in Chapter 1 of designing methods for the computation of order-independent characteristics for the entire class of context-free languages the interest is now focussed on the *expansive context-free grammars*. It is hypothesised that a method indirectly using a regular language representation of the stochastic language exists, and this chapter is devoted to the verification of that hypothesis. The method makes highly use of a power series representation of the language, and a short introduction to formal power series is therefore given.

4.1 The hypothesis

The main statement of the hypothesis is that it is possible for any stochastic expansive context-free language to compute the set of order-independent statistical characteristics pointed out in Chapter 1. Furthermore, this can be done by utilizing a simplification of the computations equivalent to the one obtained from a regular language representation. The simplification comes from letting the symbols commute and from working on the commutative map of the context-free language.

Like in Chapter 3 the hypothesis involves transforming the stochastic grammar into a non-stochastic derivation grammar. The transformed grammar will remain context-free and expansive. It can then be represented by a set of functional equations. It is known that expansive context-free grammars can be mapped into a special form of functional equation for which *Lagrange's Inversion Formula (LIF)* [27] can provide a power series representation of the language. LIF is defined for the one variable case (one variable = one nonterminal), and it is a part of the hypothesis that the generalisation of LIF by Good [23], which will here be called GLIF, can be used to give a power series representation of a language represented as a grammar with multiple nonterminals.

The power series representations will turn out to be a very suitable representation of the language in terms of computing statistical characteristics, because it contains the correct count of the different derivation trees of the individual strings.

In the process of deriving the power series solution, both LIF and GLIF assume commutativity of the variables, and the solution contains no information about the ordering of the symbols. The solution will be viewed as a indirect regular language representation due to the commutativity.

The above mentioned approach can be stated in a more precise manner by the following hypothesis.

Order-independent characteristics for stochastic expansive context-free languages can be computed using the following three-step-procedure:

1. *Change the stochastic context-free grammar into a non-stochastic derivation grammar.*
2. *Represent the derivation grammar as a set of functional equations. Represent the result in terms of power series by use of the Generalized Lagrange Inversion Formula, (or by use of LIF).*
3. *Give an interpretation of the power series in terms of statistical characteristics.*

The following sections will verify the hypothesis.

4.2 Step one

In step one the stochastic expansive grammar is changed into a non-stochastic derivation grammar. The situation is completely identical to the one in Chapter 3, so reference is made to Section 3.2.

4.3 Step two

Chomsky and Schützenberger [8] have shown that a context-free language can be represented as a power series. In this section their ideas will be outlined, followed by two methods for obtaining a regular language representation of the expansive language. One method is for grammars with only one nonterminal and is described in order to establish the foundation, on which the generalization to multiple variables can be build.

4.3.1 Formal power series for context-free languages

Any context-free grammar can be represented as a set of equations, for which a solution can be shown to exist [8]. The solution can be expressed as a power series where the coefficients represent the number of different ways to derive a given string [8]. Through a series of definitions and theorems this section outlines their ideas as they have been further developed and formulated by

Saloma and Soittola [42], Kuich and Salomaa [30], and Salomaa [41]. Both the case with non-commutative variables and the one with commutative variables are treated. The outline starts with a few definitions concerning algebraic systems leading to the definition of a formal power series.

An algebraic system consists of a set M and some operations $o_1, o_2, \cdots, o_l$ defined on the set M. The following notation is used $\mathcal{M} = < M; o_1, o_2, \cdots, o_l >$. A monoid is an example of a very simple algebraic system.

Definition 17 *A* **monoid** *is an algebraic system $< M, \star >$where $\star$ is a binary associative operation. It has a unique neutral element denoted e, so that $a \star e = e \star a = a$, $a \in M$*

Another algebraic system of interest to the formulation of power series is the semiring.

Definition 18 *A* **semiring** *is a system$< A; +, \cdot >$ where,*

1. *$< A; + >$ is a commutative monoid with 0 as the neutral element for addition.*
2. *$< A; \cdot >$ is a monoid with 1 as the neutral element for multiplication.*
3. *Multiplication is distributive over addition.*
4. *$0 \cdot a = a \cdot 0 = 0$ for every $a \in A$.*

As examples of semirings can be mentioned $\mathcal{N}$, $\mathcal{I}$ and $\mathcal{C}$, denoting the semirings of the natural numbers, integers and complex numbers respectively. Combining the two definitions it is possible to define a formal power series.

Definition 19 *A* **formal power series r** *is any mapping from $M^\star$ into a semi-ring $\mathcal{A}$.*
r will be an infinite sum of the form $r = \sum_{w \in M^\star} (r, w) w$

The term (r, w) will be called the coefficients of the series, and r is a series with *non-commuting variables* in M. The set of all power series based on M and A is denoted by $A << M >>$.

Definition 20 *For a given power series r in $A << M >>$ the set of strings defined as*

$$\{w \mid (r, w) \neq 0\}$$

is **the support of r** *and is denoted by supp(r).*

A monoid of special interest in language theory is the *free monoid* $\Sigma^\star$, defined over an alphabet Σ and for the operation called catenation. The neutral element is the empty string λ. $\Sigma^\star$ represents all possible combinations of elements from the alphabet. As stated in Chapter 2 any subset of $\Sigma^\star$ is a language. Consider now a function r which transforms any element in $\Sigma^\star$ into a number representing a property of the element. The mapping could be as simple as: assign 1 to w

if $w \in \Sigma^\star$ **and** $w \in L$, otherwise assign 0. The language expressed in terms of the power series is identical to $supp(r)$.

A grammar G generating the context-free language $L(G)$ can be represented as a set of equations for which a solution exists.

Definition 21 *Consider a context-free grammar $G = (N,\Sigma,P,S)$ where $N = \{A_1, A_2, \ldots, A_m\}$, $\Sigma = \{a_1, a_2, \ldots, a_k\}$, and a set of productions of the form $A_i \to \beta$, where $\beta \in (N \cup \Sigma)^\star$. An* **algebraic system of equations** *is a set of equations representing G having the form:*

$$\begin{aligned} A_1 &= f(A_1, A_2, \ldots, A_m, a_1, a_2, \ldots, a_k) \\ A_2 &= f(A_1, A_2, \ldots, A_m, a_1, a_2, \ldots, a_k) \\ &\vdots \\ A_m &= f(A_1, A_2, \ldots, A_m, a_1, a_2, \ldots, a_k) \end{aligned}$$

If the grammar does not contain productions of the form $A_i \to \lambda$ or $A_i \to A_j$ the system of equations is termed **proper**.

Theorem 10 *For each proper algebraic system of equations defined as above, there exist exactly one solution $\sigma = (\sigma_1, \ldots, \sigma_m)$ for which the empty string is not included.*

There is one solution for the entire set of equations made up of solutions for each of the nonterminals. When modelling a grammar the only interesting item is the solution related to the nonterminal, which is the starting symbol. The solution for a context-free grammar can be expressed as a power series as shown by the following definition and theorem.

Definition 22 *A power series r in $A << \Sigma >>$ for which $(r, \lambda) = 0$ is termed* **A-algebraic** *iff it is a component of the solution of a proper algebraic system of equations.*

Theorem 11 *Let there be given a context-free grammar G defining a language L(G), then there exist a N-algebraic series r, for which supp(r) equals L(G).*

As N is the set of natural numbers {0,1,2,3, …} the theorem says that any context-free language can be represented as a series with coefficients from {0,1,2,3, …}. The interpretation of these coefficients in language theoretical terms is determined through the next theorem.

Theorem 12 *The coefficient of each word in the N-algebraic series r(G) generated by a context-free grammar G equals the number of different derivations of w.*

If the coefficient is 0 the string is not in the language, if it is 1 the string is in the language and can be generated unambiguously, if it is 2 two different derivations trees for the same string exist, etc..

Changing a grammar from having non-commutative symbols to one with commutative symbols will induce some changes on the language generated by the grammar. These changes can be described as a mapping carrying all strings having identical commutative images into one string. The sum of different derivations for this commutative string will then be the sum of the derivations for the individual trees represented by the string in the commutative language. Such changes do, of course, not effect any of the fundamental theorems just outlined. It requires, however, a definition of a commutative formal power series.

Let $c(M^\star)$ be the *commutative monoid.* Assume that $M = \{x_1, x_2, \ldots, x_m\}$, then $c(M^\star)$ can be written as

$$c(M^\star) = \{x_1^{n_1} \cdots x_m^{n_m} \mid n_1, n_2, \ldots, n_m \geq 0\}$$

Definition 23 *A* **commutative formal power series c(r)** *is any mapping from $c(M^\star)$ into a semiring $\mathcal{A}$. c(r) will be an infinite sum of the form*

$$c(r) = \sum_{n_1,\ldots,n_m \geq 0} (r, x_1^{n_1} \cdots x_m^{n_m}) x_1^{n_1} \cdots x_m^{n_m}$$

The two definitions concerning power series are related in the following way:

$$c(r) = \sum_{n_1,\ldots,n_m \geq 0} \sum_{c(w)=x_1^{n_1},\ldots,x_m^{n_m}} (r,w) x_1^{n_1} \cdots x_m^{n_m}$$

When the commutative monoid is the commutative free monoid $\Sigma^\star$ the theorems from the non-commutative case can be combined to the following theorem:

Theorem 13 *Let G be a grammar generating a context-free language L. Let c(R(G)) be a mapping from $c(\Sigma^\star)$ into the semiring $\mathcal{N}$. Then the coefficients $(r, x_1^{n_1} \cdots x_m^{n_m})$ in c(r(G)) equal the number of different derivation trees in L, having identical commutative maps.*

Proof:
Follow directly from the definition of (r, w) for the non-commutative case and from the definition of $c(r)$ for the commutative case.
End of proof.

From the field of complex analysis the following definition of a power series can be found [29],

Definition 24 A power series in powers of z-a *is an infinite series of the form:*

$$\sum_{m=0}^{\infty} c_m (z-a)^m = c_0 + c_1(z-a) + c_2(z-a)^2 + \cdots$$

where z is a variable, $c_0, c_1, \ldots$ are coefficients and a is the center of the series.

A power series for multiple variables can be obtained in a similar way by treating m, c, z and a as vectors. Setting $a = 0$, power series in powers of z can be obtained.

$$\sum_{m=0}^{\infty} c_m z^m = c_0 + c_1 z + c_2 z^2 + \cdots$$

It can be shown that any analytic function can be represented by such a power series. This means that for a given value of z, the sum of terms on the right hand side converge to the value of $f(z)$.

The following notation will be used in the remaining chapters of this thesis. Let $r(x)$ be a power series in powers of x then $[x^n]r(x)$ denotes the coefficient of x_n in power series $r(x)$.

To differentiate between the two definitions of power series given in this section the name formal is attached to the series relating to letters from an alphabet.

There is a close relationship between the two representations which will be discussed and used extensively in the following sections.

4.3.2 Obtaining power series using LIF

It will now be investigated how a power series representation of a language can be obtained. The presentation will be divided into two parts. In this section grammars with only one nonterminal will be treated, and in the following section the generalization to multiple nonterminals are stated.

The non-stochastic derivation grammar for any stochastic context-free grammar having only one nonterminal will have productions of the following general form,

$$S \to a_i S^i$$

i.e., it will consist of a finite number of the following productions.

$$S \to a_0, S \to a_1 S, S \to a_2 S^2, \ldots$$

This can be recognized as the Lukaciewicz languages [4]. The languages can be characterized in the following way. Define an injective mapping $\delta : \Sigma^\star \to I$, mapping any terminal symbol of the grammar into an integer in the following way,

$$\delta(a_i) = i - 1$$

and mapping a sequence of terminals,w, into a number by,

$$\delta(w) = \sum_{a_i \in w} \delta(a_i)$$

Definition 25 A Lukaciewicz language *is the set of words from* $\Sigma^\star$ *such that*

- $\delta(w)$ *= -1, and*
- $\delta(w') \geq 0$*, where* w' *is any prefix of* w*.*

Lukaciewicz languages can be represented as an equation on the following form

$$S = a_0 + a_1S + a_2S^2 + a_3S^3 + \cdots$$

which in general is a functional equation of the form $S = f(S)$. A solution to the functional equation is any string derivable from the initial grammar, i.e., any string in the Lukaciewicz language. The set of all solution is the language itself.

To obtain a representation of the commutative map of the language a mathematical tool called Lagrange's Inversion Formula (LIF) [50, 24] is used. It is very useful in obtaining solutions to functional equations of the form,

$$z = \xi f(z)$$

The solution will be expressed in terms of power series in powers of t. The equation of the Lukaciewicz language can easily be changed to fit the requirements of LIF, simply by introducing a dummy variable t which will then be used as a place-holder for the results of LIF, but given no language theoretical interpretation. It then becomes,

$$S = tf(S)$$

LIF can be expressed in the following way, using the formulation by Good.

Lagrange's Inversion Formula *Let z= $\xi f(z)$, where f is analytic in a neighbourhood of the origin and does not vanish there, and suppose h(z) is also analytic in a neighbourhood of the origin. Then z can be expressed in a power series in ξ and*

$$[\xi^m]h(z) = m^{-1}[z^{m-1}](h'(z)f(z)^m)$$

where m = 1,2,3,....

LIF is stated for complex analytic functions, and the reader is referred to Harrison [27] for a strictly mathematical example, i.e., $x = ye^x$. In this thesis a language theoretical interpretation of LIF will be used. A simple example is given in Harrison, and a more general discussion has been given by Lothaire [33] and Raney [39]. The language theoretical interpretation is based on the following relationship between LIF and the context-free grammar G.

- The equation $f(S)$ obtained from the grammar is the function $f(z)$ in LIF. The nonterminal of the grammar becomes the variable in LIF.
- A dummy variable t is inserted in the equation $f(S)$, but is given no language theoretical interpretation.
- For the function $h(z)$ in LIF the following function will be used, $h(S) = S$.

As can be seen directly from the formula, LIF has the potential of solving the equation and expressing the solution, not just for u, but for any analytic function of z, $h(z)$. This will, however, not be used in this formal language application of the formula.

Once the relationship between LIF and the grammar G is known, it is possible to evaluate, if the requirements of LIF is fulfilled by the formal language. The requirement is that $f(S)$ and $h(S)$ are analytic functions. Previously it was argued that when a Lukaciewicz language is represented as an equation it will take the form,

$$S = a_0 + a_1 S + a_2 S^2 + a_3 S^3 + \cdots$$

which is a polynomial in S. It is well known that that any polynomial is an analytic function, e.g., see Colombo [10]. $h(S)$ can be viewed as a constant polynomial, and therefore both requirements are fulfilled.

In an attempt to relate the computations performed by LIF to operations on L, the following theorem from Lothaire [33] based on the work by Raney [39] is of interest.

Theorem 14 *Represent a Lukaciewicz language as a functional equation:*

$$S = tf(S)$$

then a unique power series solution ξ exists so that

$$n[t^n]S = [S^{n-1}]f(S)^n$$

Proof:
An outline of an combinatorial proof is given, the reader is referred to Lothaire or Raney for full details.

Define a mapping U from $\Sigma^\star$ into an algebraic field K so that

- $U(a_i) = u_i$, where $a_i \in \Sigma$.
- $U(a_i \star a_j) = U(a_i) \cdot U(a_j)$, where $\star$ is catenation and $\cdot$ is multiplication.
- $U(B) = \sum_{w \in B} U(w)$, where B is a finite subset of $\Sigma^\star$.

Basically the mapping generalize the situation by taking the problem from a specific algebraic system with the catenation-operation to a more general system with multiplication and addition.

It is argued that the unique solution S of the functional equation has the following property

$$[t^n]S = U(L \bigcap \Sigma^n)$$

The coefficient to t^n in S is related to the words of length n in the Lukaciewicz language L.

Let $\delta^{-1}(-1)$ be all strings w in L having $\delta(w) = 1$, i.e., all strings fulfilling the first of the two conditions for strings in a Lukaciewicz language. It can be shown that for each string in L, there are n strings in $\Sigma^\star$ with $\delta^{-1}(-1)$, which can be expressed as

$$n[t^n]S = U(\Sigma^\star \bigcap \delta^{-1}(-1))$$

By looking at the outcome when the polynomial $f(S)$ is raised to power n the following observation can be made,

$$U(\Sigma^\star \bigcap \delta^{-1}(-1)) = [S^{n-1}]f(S)^n$$

By combining the statements the required result is obtained.
End of Proof.

Based on the description so far in this chapter it is possible to conclude the following:

A context-free grammar G, generating $L(G)$, can be represented by a set of algebraic equations. The equations have a solution which can be expressed as a formal power series $r(L)$. The commutative map of L can be represented likewise in terms of a commutative formal power series, $c(r(L))$.
For the one nonterminal case G can be mapped into a functional equation, for which a solution, representing the commutative map of L, can be obtained in terms of a power series.
The commutative formal power series and the power series both representing the commutative map of L are identical, and the relationship between them can be expressed as

$$[t^n]S = (r,w)w \qquad \text{for } (r,w) \neq 0 \wedge |w| = n$$

To illustrate the language theoretical interpretation of the result from LIF the following grammar will be used. $G_{LIF} = (\{S\}, \{a, b\}, P, S)$ where P contains the two productions; $S \to b, S \to aSS$. Written as an equation the productions look like $S = b + aS^2$. Using LIF the result becomes:

$$S = bt + abbt^3 + 2a^2b^3t^5 + 5a^3b^4t^7 + \cdots$$

If the productions are used, the following strings can be generated:

$$L = \{b, abb, aabbb, ababb, aaabbbb, aababbb, aabbabb, abaabbb, abababb, \ldots\}$$

The commutative map of L, $\Phi(L)$ becomes:

$$\Phi(L) = \{b, ab^2, 2a^2b^3 + 5a^3b^4 + \cdots\}$$

which is identical to the results generated using LIF when t is considered a dummy variable.

Some insight into how LIF work on languages can be obtained by looking at how LIF process $f(S)$. Consider as an example the case where the information related to strings of length 5 is obtained. For $f(S) = b + aS^2$ strings with five symbols come from $(b + aS^2)^5$. This expand into a total of 32 terms including the following:

$$\begin{aligned}(b + aS^2)^5 = \quad & bbbbb + bbbbaS^2 + bbbaS^2b + bbbaS^2aS^2 + \cdots + aS^2baS^2bb + \cdots \\ & + aS^2a^2bbb + \cdots + aS^2aS^2aS^2aS^2aS^2\end{aligned}$$

Out of the 32 terms only 2 represent valid strings from L. Only the 10 terms containing S in power $n - 1$, i.e., S^4, will have the right combinations of a's and b's to fulfill the requirement of having $\delta(w) = -1$. The coefficients of those 10 terms are listed below;

bbbaa	*abbba*
bbaba	*abbab*
bbaab	*ababb*
babba	*aabbb*
babab	
baabb	

These can easily be identified as all combinations of 2 a's and 3 b's. They are a result of purely combinatorial considerations without any relation to the restrictions on the ordering of the symbols put forward by the productions. Referring to the previous listing of strings in L it can be seen that only the last 2 coefficients in the above list for $(b+aS^2)^5$ are valid strings. Only those two can be constructed from the productions, i.e., they are the only strings that fulfill both of the requirements for Lukaciewicz languages. The last requirement is that any prefix w' of the string w should have $\delta(w') \geq 0$. As an example *bbbaa* has proper prefix; λ, b, bb, bbb, $bbba$. By definition $\delta(\lambda) = 0$, but $\delta(b) = -1$, $\delta(bb)$ = -2, $\cdots$. The valid string *ababb* has proper prefix; λ, a, ab, aba, $abab$, with $\delta(\lambda)$ =0, $\delta(a)$ =1, $\delta(ab)$ = 0, $\delta(aba)$ =1, $\delta(abab)$ = 0.

4.3.3 Obtaining power series using GLIF

LIF can only be used for expansive grammars having one nonterminal, and it is therefore necessary to look for a more versatile tool for handling all expansive grammars. In this section a generalization of LIF due to Good is introduced and its applicability to expansive grammars is documented.

A context-free grammar $G = (N, \Sigma, P, S)$ will in general make use of a set of nonterminals $N = \{A_1, A_2, \ldots, A_m\}$ and a set of productions of the form $A_i \to \beta$, where $\beta \in (N \cup \Sigma)^\star$. The set of all productions can be partitioned into equivalence classes C_i, one for each nonterminal. C_i contains all productions having A_i as its left-hand side. The grammar can be represented as a system of equations where each equation will be on the following general form:

$$A_i = \beta_{i_1} + \beta_{i_2} + \cdots + \beta_{i_l}$$

$\beta_{i_1}, \ldots, \beta_{i_l}$ are all the individual right hand sides in C_i. In the equations the terminals become constants, and the nonterminals play the role of variables. For a grammar with k nonterminals it can be represented as a system of k equations as described in Definition 21.

The complete solution for equation $A_i = f(A_1, A_2, \ldots, A_n)$ is the entire set of strings from $\Sigma^\star$ which can be derived using the productions from P, having started the rewriting process with nonterminal A_i. Let A_S be the starting symbol of the grammar. Then only the equation for A_S will be of direct interest since only that solution will be identical to the language generated by G, $L(G)$.

Due to the success of using LIF for the one-variable case it seemed natural to look for a generalization of LIF in order to derive a power series representation of the system of equations. Such a generalization exists. It was developed by Good [23] in 1960. Following the ideas of Good, GLIF can be stated as,

The Generalization of Lagrange's Inversion Formula (GLIF)
Let $\mathbf{z} = (z_1, z_2, \ldots, z_n)$, $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_n)$, $\mathbf{f} = (f_1, f_2, \ldots, f_n)$. *Consider a system of equations each being on the form:*

$$z_i = \xi_i f_i(\mathbf{z})$$

If each component of $\mathbf{f}(\mathbf{z})$ *is analytic at the origin and does not vanish there, and* $G(\mathbf{z},\boldsymbol{\xi})$ *is meromorphic in the neighbourhood of* $\mathbf{z} = \boldsymbol{\xi} = \mathbf{0}$, *then*

$$[\boldsymbol{\xi}^{\boldsymbol{m}}]G(\mathbf{z}(\boldsymbol{\xi}), \boldsymbol{\xi}) = [\mathbf{z}^{\mathbf{m}}]\left\{ G(\mathbf{z}, \boldsymbol{\xi}(\mathbf{z}))\mathbf{f}(\mathbf{z})^{\mathbf{m}} \left\| \delta_i^j - \frac{z_i}{f_i(\mathbf{z})} \frac{\partial f_i(\mathbf{z})}{\partial z_j} \right\| \right\},$$

where $\|a_{ij}\|$ *denotes the determinant of the matrix* (a_{ij}) *and where i,j = 1, 2,* $\cdots$*, n, and the components of* $\mathbf{m}$ *are integers, possibly negative. The notation implies that G is to be regarded as a function of* $\boldsymbol{\xi}$ *on the left of the equation, and as a function of* $\mathbf{z}$ *on the right.*

Similar to the interpretation for LIF it is possible to relate GLIF to languages represented as grammars. This is done in the following way,

- Each of the equations of the form $A_i = f(A_1, A_2, \dots, A_n)$ is augmented by a dummy variable t_i to take the form $A_i = t_i f(A_1, A_2, \dots, A_n)$. Each new augmented equation will be used as an equation $z_i = \xi_i f(\mathbf{z})$ in GLIF. The nonterminals in the grammar thereby become the variables in $f(\mathbf{z})$. The terminals do not contribute to the derivation process and can therefore be viewed as constants.

- The function $G(\mathbf{z}(\boldsymbol{\xi}), \boldsymbol{\xi})$ in GLIF has the role of specifying which relationship between the variables of the system, should be expressed by the resulting power series. From a language theoretical point of view the only solution of interest is the one for the equation for the starting-symbol A_S. The function $G(\mathbf{z}(\boldsymbol{\xi}), \boldsymbol{\xi})$ will therefore be replaced with A_S.

It has not been possible to give a combinatorial proof of the generalization from which documentation of its relation to words(formal language) could have been taken. Documentation of GLIF's applicability to formal languages is instead based on proving that the interpretation fulfills the requirements put forward by GLIF.

The requirements are that each $A_i = f(A_1, A_2, \dots, A_k)$ is analytic and that A_S is meromorphic. Each of the equations are polynomials in several variables. It is well-known that any polynomial in several variables is an analytic function, e.g., see Range [40]. A meromorphic function is any function for which it can be stated that if the function has singularities they are poles [29]. A singularity at a given point in the plane arises when the function is not analytic in that point. For a singularity to be a pole the Laurent series representing the function in the singularity point must contain only a finite number of terms in its principal part (terms of the form $c_n/(z-a)^n$). Since A_S can be viewed as a constant polynomial it is analytic over the entire plane, and both requirements are therefore fulfilled.

To develop an understanding of how GLIF works on languages, some of the differences between LIF and GLIF are considered. The differences relate to how the methods process the equations from the grammars, and how the counting problem look like in terms of derivation trees.

In terms of a language theoretical interpretation the formulation of GLIF must comprise 3 different mechanisms,

1. Provide all possible combinations of the productions of the grammar.

2. Point out the set of valid strings among all the combinations.

3. Provide the correct coefficients, i.e., the count of different derivation trees for all strings having identical commutative maps.

The first mechanism is obtained by $\mathbf{f}(\mathbf{z})^{\mathbf{m}}$. In a more expanded form it looks like

$$f_1^{m_1}(\mathbf{z}) \cdot f_2^{m_2}(\mathbf{z}) \cdots f_n^{m_n}(\mathbf{z})$$

The integer m_i represents the number of times productions from equivalence class C_i will be used. If in general the subset of $L(G)$ consisting of all strings using exactly k productions from G is of interest, then the following combinations should be considered:

$$\sum_{m_1+m_2+\cdots+m_n=k} f_1^{m_1} \cdot f_2^{m_2} \cdots f_n^{m_n}$$

The sum is for all combinations of $m_1, \ldots, m_n$ so that the sum equals k.

The set of valid strings is determined by

$$[\mathbf{t}^{m_1,\ldots,m_n}] = [A_1^{m_1} \cdots A_n^{m_n}]S \cdot f_1(N)^{m_1} \cdots f_n(N)^{m_n}$$

where N is the set of nonterminals. While computing $S \cdot f_1(N)^{m_1} \cdots f_n(N)^{m_n}$ a large number of terms is produced. Each term will during the enumeration process have picked up specific amounts of variables. The number of variables in a term equals the number of productions which must be used to rewrite all nonterminals in all of the sentential forms for the derivation process of the string. Therefore only terms having A_i^q in $\mathbf{f}(N)^{\mathbf{m}}_{|m_i=q}$ can be valid strings. A special situation exists for the starting symbol A_S. The rewriting process is started by one occurrence of the starting symbol. This is not introduced through any production and is therefore terms with A_S^{q-1} in $\mathbf{f}(N)^{\mathbf{m}}_{|m_S=q}$ which denote valid strings. This is identical to what was observed for LIF. This can be realized by multiplying $f(A_S)^{m_S}$ with S and using the terms with $A_S^{m_S}$.

As an example consider the grammar $G_{GLIF} = (\{A, S\}, \{a, b, c, d\}, P, S)$ where P:

$$\begin{array}{ll} S \to aASS & A \to cAA \\ S \to b & A \to d \end{array}$$

Consider the case of $m_S = m_A = 3$:

$$\begin{aligned} (aAS^2 + b)^3 &= a^3S^6A^3 + 3a^2bS^4A^2 + 3ab^2S^2A + b^3 \\ (cA^2 + d)^3 &= c^3A^6 + 3c^2dA^4 + 3cd^2A^2 + b^3 \end{aligned}$$

None of the terms resulting from the above calculations are valid strings, i.e, no terms have S^3 in $f_S(N)^3$ or A^3 in $f_A(N)^3$, but by computing $S \cdot f_S(N)^3 f_A(N)^3$ the following result appears:

$$S \cdot f_S(N)^3 f_A(N)^3 = a^3c^3S^7A^9 + \cdots + ab^2cd^2S^3A^3 + \cdots b^6S$$

where $ab^2cd^2S^3A^3$ is a valid string.

The mechanism for computing the correct coefficient must take into consideration that the count of different derivation trees is strongly dependent on how the individual productions are linked together. The grammar used above can also be used to describe the derivation process which must be modelled by GLIF. Remember that in the previous section it was stated that a grammar

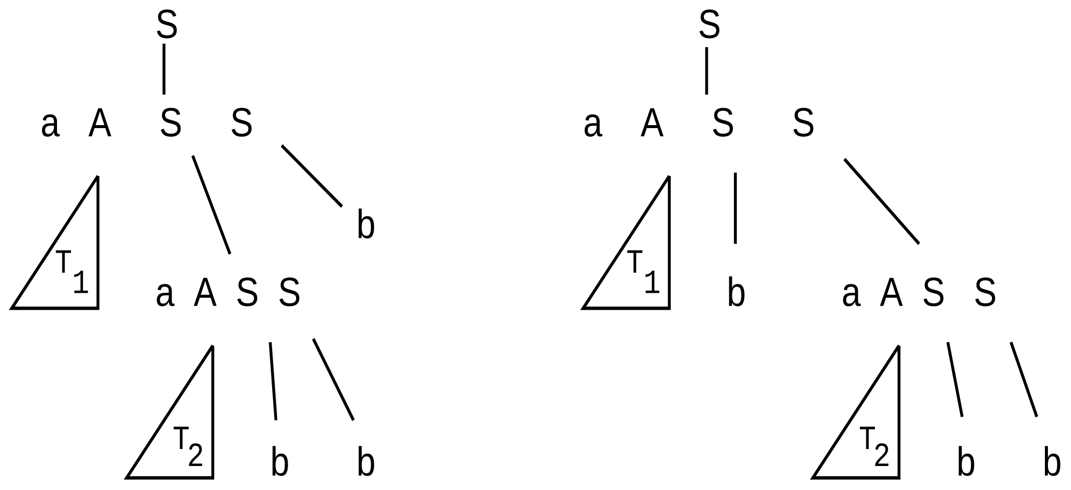


Figure 4.1: The basic structure of the derivation trees for strings with commutative map $a^2b^3c^2d^4$

$G = (\{A, B\}, \{a, b\}, P, S)$ with the equation $S = aSS + b$ would give rise to the power series:

$$S = bt + ab^2t^3 + 2a^2b^3t^5 + \cdots$$

Note that there will be two different trees having two a's and three b's.

The power series for G_{GLIF} becomes:

$$S = bt_S + ac^2dt_S^3t_A + ab^2cd^2t_S^3t_A^3 + \cdots + 10a^2b^3c^2d^4t_S^5t_A^6 + \cdots$$

By looking at the derivation trees the count of 10 trees with yield $a^2b^3c^2d^4$ can be explained. In Figure 4.1 two basic derivation trees for the 10 trees are shown.

The basic trees reflect the structure of the two trees from G_{LIF} for a^2b^3. In each of the two trees two occurrences of nonterminal A exist. From each of those a subtree can develop. Call the trees T_1 and T_2. The occurrences of two c's and four d's can be distributed between the two trees in different ways. Since no nonterminal are attached to the d's only the c's are of interest. They can be distributed with two in T_1 and zero in T_2 or vice versa, or there can be one in each of the subtrees. Whenever 2 c's occur in the same tree they can themselves be configurated in two ways (according to G_{LIF}). For each basic tree the count becomes $2 + 1 + 2 = 5$. The total count becomes $2 \cdot 5 = 10$.

If the dependencies between the productions are increased by changing the production $S \rightarrow aASS$ to $S \rightarrow aAASS$ then the series becomes:

$$S = bt_S + ab^2d^2t_S^2t_A + \cdots + 28a^2b^3c^2d^6t_S^5t_A^8 + \cdots$$

The reason for the increase in different trees is that the two c's and six d's now can be distributed between 4 different trees.

The dependencies between nonterminals in the different productions are captured by GLIF through the determinant

$$\left\| \delta_i^j - \frac{z_i}{f_i(\mathbf{z})} \frac{\partial f_i(\mathbf{z})}{\partial z_j} \right\|$$

It has, however, not been possible to dissect the computations represented by the Jacobian determinant so that an explanation of HOW it produces the correct count can be given a language theoretical interpretation.

Despite the fact that GLIF is a generalization of LIF it does not provide a usable method for the case of only one nonterminal, due to the formulation of GLIF in terms of a determinant of a matrix.

4.4 Step three

In step 2 a power series representation should be obtained based on LIF or GLIF. The preceding sections have shown that such series can be computed for any context-free grammar, and that the coefficients of the series represent the number of different derivation trees for any specific string. The algorithm for obtaining the power series representation is given in Algorithm 6.

The only thing remaining in order to verify the entire hypothesis is to devise a way to compute statistical characteristics based on the power series representation. To do so it is important to remember that the power series represents a derivation grammar. It is therefore possible to represent any information concerning random variables of the productions for the original stochastic grammar, in the power series through the unique productions labels of the productions. Those labels are the terminals of the derivation grammar and appear in the coefficients of the series.

To compute information for a specific random variable RV it is necessary to make an inquiry into the original grammar and determine the number of occurrences that the production gave rise to of the random variable of interest. If a production having label p_i has k occurrences of the random variable then p_i should be substituted by $p_i x^k$ in the power series. Once the substitution has been performed, terms of identical powers of x can be collected and the symbol p_i can be replaced with the production probability of production p_i. Setting all t^k equal to 1, the coefficient of the powers of x can be computed. The coefficients of x_k can be interpreted as the accumulated probability mass attached to strings in L having k occurrences of the variable of interest. Algorithm 7 describes an algorithm for computing the statistical characteristics.

By combining Algorithm 6 and 7 a way has been devised for computing statistical characteristics for any stochastic context-free language. The hypothesis from Section 4.1 is therefore considered verified.

4.5 Computing characteristics

As is the case for the method for non-expansive grammars this method is also an enumerative technique. This means that the same kind of enumerative problems exists for this method. First an example is given followed by some general considerations.

4.5.1 An example

Let there be given a stochastic context-free grammar $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, where P has the productions:

$$\begin{array}{ll}
p_1 : S \rightarrow aSSA & p_4 : A \rightarrow aAB \\
p_2 : S \rightarrow Ab & p_5 : A \rightarrow bb \\
p_3 : S \rightarrow c & p_6 : B \rightarrow bABB \\
 & p_7 : B \rightarrow cc
\end{array}$$

The productions in the non-stochastic derivation grammar look like,

$$\begin{array}{ll}
S \rightarrow p_1 SSA & A \rightarrow p_4 AB \\
S \rightarrow p_2 A & A \rightarrow p_5 \\
S \rightarrow p_3 & B \rightarrow p_6 ABB \\
 & B \rightarrow p_7
\end{array}$$

A representation of the grammar as a system of equations becomes,

$$\begin{aligned} S &= p_1SSA + p_2A + p_3 \\ A &= p_4AB + p_5 \\ B &= p_6ABB + p_7 \end{aligned}$$

Augmenting the equations to fit the requirements of LIF/GLIF,

$$\begin{aligned} S &= t_S(p_1SSA + p_2A + p_3) \\ A &= t_A(p_4AB + p_5) \\ B &= t_B(p_6ABB + p_7) \end{aligned}$$

The power series representation of $L(G)$ becomes,

$$r(S) = p_3t_3 + p_2p_5t_St_A + p_2p_4p_5p_7t_St_A^2t_B + p_2p_4^2p_5p_7^2t_St_A^3t_B^2 + \cdots$$

Let the random variable of interest be the string length. By substituting the p_i's with p_ix^k where k represents the number of occurrences of symbols in a string, a new power series in x can be constructed. Setting $t_S, t_A, t_B = 1$ the series becomes;

$$r(R) = p_3x + p_2p_5x^3 + p_2p_4p_5p_7x^6 + p_2p_4^2p_5p_7^2x^9 + \cdots$$

Assume that $p_1 = 0.5$, $p_2 = 0.2$, $p_3 = 0.3$, $p_4 = 0.6$, $p_5 = 0.4$, $p_6 = 0.7$, and $p_7 = 0.3$

Based on the random variable string length it is possible to compute, e.g., the probability mass associated with the strings in $L(G)$ having a string length less than 8.

$$P(w < 8) = 0.3 + 0.08 + 0.01444 = 0.3944.$$

4.5.2 General considerations

As with all enumerative techniques there is the problem of determining the number of enumerations required to obtain a result with a given accuracy. As the previous example showed there are situations where the correct value of the characteristic can be obtained after only a few iterations of the formula. However, situations exist for which a larger number of iterations are required and there even exist situations where all strings in the commutative map should be considered. To put the problem in its right perspective it must be remembered that as the production probabilities are numbers less than 1 the probability of a string rapidly decreases as the length of the string increases. The characteristics can be determined with any precision required simply by using more and more strings for the computation. In practice it is advisable to make use of a threshold on the increase in the probability mass for the characteristic of interest as the number of iterations grows. For more information on the characteristics computable by LIF and GLIF see Chapter 5.

4.6 Summary

The hypothesis put forward in Section 4.1 has been verified. A regular language representation is shown to exist by which order-independent statistical characteristics can be computed for expansive stochastic context-free languages.. The method makes use of a transformation from the stochastic to the non-stochastic domain based on a derivation language representation. The grammar is then represented as a set of equations, and a method called The Generalized Lagrange Inversion Formula known from complex function theory is used to derive a solution of the system of equation. The result is a formal power series representation of derivations in the stochastic language and from that the statistical characteristics can be computed.

The chapter shows that the strategy for changing from the stochastic to the non-stochastic domain and back, developed in Chapter 3, also can be applied to the expansive case.

The chapter provides new insight into how methods from complex function theory can be applied to the field of language theory. It appears to be the first time the Generalized Lagrange Inversion Formula has been used to represent grammars.

By the design of the method it has become possible to use an enumerative technique for the computation of order-independent random variables, which is not based on the generation of all ordered strings in the language. Instead only those with different commutative maps will be generated.

ALGORITHM 6. Create a power series representation of the SCFL

```
Input:

Output:

Method:
A stochastic context-free grammar $G = (N, \Sigma, P, A_S)$.

A power series, $r(S)$ of $L(G)$.

1)  Apply Algorithm 1 to get a non-stochastic derivation grammar $DG$.

2)  Apply Algorithm 2 to represent $DG$ as a set of equations.

    Augment each equation from
3)      $A_i = f(A_1, \ldots, A_n)$
    to
        $A_i = t_i f(A_1, \ldots, A_n)$

    If number of nonterminals in $N = 1$ then
4)  For m = 0 to Infinity do
        $[t^m]r(S) = m^{-1}[A_S^{m-1}] f_S(N)^m$
    End

    If number of nonterminals in $N \geq 2$ then
5)  Compute $\Delta = \left\| \delta_i^j - \frac{z_i}{f_i(A_1 \cdots A_n)} \frac{\partial f_i(A_1 \cdots A_n)}{\partial z_j} \right\|$
    where $i, j = 1...n$.

    For $(m_1, m_2, \ldots, m_n) = 0$ to Infinity do

        $[t^{m_1,\ldots,m_n}]r(S) = [A_1^{m_1} \cdots A_n^{m_n}] A_S \cdot f_1(N)^{m_1} \cdots f_n(N)^{m_n}$

    where $(m_1 \cdots m_n) = k$ denotes all combinations of $m_1$ to
    $m_n$ so that their sum is $k$. The coefficients will be sequences
    of terminals from the derivation grammar.
    End
```

ALGORITHM 7. Computing statistical characteristics based on a power series representation of a SCFL.

Input:

Output:

Method:

A random variable RV.
A stochastic context-free grammar $G = (N, \Sigma, P, S)$.
A power series $r(S)$ representing a SCFL. The coefficients of $r(S)$ are sequences of production labels p_i for productions from G.

A power series $r(RV)$ in powers of x. The coefficient to x^k is the probability of k events of the variable RV.

```
1)    /* Insert the stochastic information */
      **For** each production in $P$ **do**
          Set $k$ equal to the number of occurrences of the random
          variable $RV$ associated with the production.
          Substitute $p_i$ with $p_i x^k$ in $GF$
      **End**

2)    Collect all terms of identical powers of $x$.

3)    Substitute the symbol $p_i$ with the probability of production $p_i$
      and compute the coefficients of the new power series $r(RV)$.
```

Chapter 5

Existing and new methods – An overview

As pointed out in Chapter 1 methods exist for computing a variety of characteristics of a stochastic context-free language. The derivation process of stochastic linear and context-free languages can be modelled by a Markov chain and a multi-type branching process, respectively. Based on these models statistical characteristics can be computed. For both models a large variety of information can be computed. This chapter does not claim completeness in the description of what can be computed for the models. Emphasis has been put on the general computations with relevance for the application to stochastic languages. To put the methods developed in this thesis in perspective this chapter gives a very brief outline of these models and their use in relation to grammars. For more details about the models the reader is referred to Appendix A and B.

The methods developed in Chapter 3 and Chapter 4 will be given a common name: The Parikh-Thomason-Larsen approach to computing statistical order-independent characteristics for stochastic context-free languages, in short the PTL-approach.

After the outline of the 3 methods now available an overview will be given of which statistical characteristics can be computed.

5.1 The Markov chain model of stochastic linear languages

A *linear* grammar $G = (N, \Sigma, P, S)$ has productions of the following forms, $A \to aAb$, $A \to a$, for $A \in N$ and $a, b \in \Sigma$. A subset of the linear grammars is the set of regular grammars. For a linear grammar each application of a production-rule introduces at most one new nonterminal in the sentential form. Therefore each sentential form leading to a terminated string contains exactly one nonterminal. The sequence of such nonterminals beginning with the starting symbol describes the derivation process of a string in the language. Such a sequence can be modelled by a Markov chain [22, 44, 46]. A more detailed description is given in Section A.1.

The derivation process of the grammar relates to the Markov chain in the following way; one state exists for each nonterminal in the grammar. Furthermore a special final state, F, is added to represent the termination of a derivation. Starting with the start state, S, which relates to the starting symbol of the grammar, transitions can be made between the individual states representing, that one nonterminal is being rewritten as another nonterminal, or the derivation process has ended with a string of only terminals. The states are connected together and a transition probability exists reflecting the possible rewritings of the grammar [46].

The Markov chain capable of modelling the derivation process can be characterised as a discrete parameter, time homogeneous, finite state stochastic process having 1st order Markov property.

5.1.1 Converting from a grammar to a Markov chain

To model the derivation process of a language $L(G)$ generated by a grammar G, it is necessary to convert the grammar into a Markov chain. The information required to specify the chain is a set of states $\{X_n\}$, a probability mass function for the start configuration and a conditional probability mass function [36, 15]. Appendix A provides an algorithm for the conversion and gives examples of this.

The probability mass function

$$p_j(n) = P[X_n = j]$$

denotes the unconditional probability of being in state j after n transitions. $p_j(0)$ denotes the probability of starting in state j. If there are n states a n-dimensional initial probability vector $\mathbf{p}(0)$ can be constructed,

$$\mathbf{p}(0) = (p_1(0), p_2(0), \ldots, p_n(0))$$

The conditional probability mass function is defined as

$$p_{j,k}(m,n) = P[X_n = k \mid X_m = j]$$

$p_{j,k}(m,n)$ gives the probability of being in state k after n transitions conditioned on being in state j after m transitions. If the chain is stationary, the notation is $p_{j,k}(n)$. For $n = 1$ it is called the *one-step transition probability.* For notational simplicity $p_{j,k}(1)$ will be written $p_{j,k}$.

If there are n states in the chain, there will be $n \times n$ one-step transition probabilities. They can be arranged in a matrix $\mathbf{P}$ so that each entry in $\mathbf{P}(1)$ is a one-step transition probability. For a Markov chain with state space {1,2,...,n} the matrix looks like

$$\mathbf{P}(1) = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & & \\ \vdots & & & \\ p_{n,1} & \cdots & \cdots & p_{n,n} \end{pmatrix}$$

The matrix is called the *transition matrix.*

The transition matrix can easily be established by inspection of the linear grammar. Let the set of nonterminals be $\{A_1, \ldots, A_n\}$ then the set of states become $\{A_1, \ldots, A_n, F\}$. For all productions on the form, $p_i : A_i \rightarrow a_k A_j a_m$ update $P_{i,j}$ with p_i. For all productions of the form, $p_i : A_i \rightarrow a_k$ update $P_{i,F}$ with p_i.

The Markov chain model contains information about the nonterminals. Since one nonterminal might, in different productions, introduce different terminals, it will become necessary to augment the chain, if explicit information about the terminals should be available. The augmentation should ensure a unique link between states and terminals in the grammar.

All information about the status of the process after n transitions is stored in the unconditional probability vector

$$\mathbf{p}(n) = (p_1(n), p_2(n), ..., p_m(n))$$

which can be computed as

$$\mathbf{p}(n) = \mathbf{p}(0)\mathbf{P}(n)$$

The set of n-step transition probabilities can be determined by multiplying the one-step transition matrix with itself n times.

$$\mathbf{P}(n) = \mathbf{P}^n$$

A state can be classified as either recurrent or non-recurrent. If it is recurrent the probability of ever returning to the state is one. The final state is recurrent. If the probability is less than one, it is a non-recurrent state.

Define m_j to be the *mean time to absorption* given that the chain started in state j. m_j can be computed as the solution to the set of linear equations:

$$m_j = 1 + \sum_{k \in T} p_{j,k} m_k$$

for $j \in T$, the set of non-recurrent states. By letting $\mathbf{m}$ denote a column vector whose probabilities are m_j the mean time to absorption can be computed as:

$$\mathbf{m} = \mathbf{N1}$$

where $\mathbf{N}$=$(\mathbf{I} - \mathbf{Q})^{-1}$, and $\mathbf{Q} = \{p_{j,k}\}$ for $j, k \in T$. $\mathbf{N}$ is called the *fundamental matrix* of an absorbing Markov chain. The element in the matrix $n_{i,j}$ is the mean number of occurrences of state j assuming that the process started in state i.

5.2 Characteristics based on the Markov chain Model

Having a Markov chain representation of the grammar and using well-known properties of the chain it is possible to determine order-independent single entity characteristics both for the derivation process and the language, without ever generating one string. Algorithm A.4 in Appendix A specifies the details of the computation. The mean string length, mean number of occurrences of terminals and nonterminals can be computed, by looking at the mean number of times the chain occupies the individual states. The information can be obtained by computing the fundamental matrix of the chain [22, 46]. Each of the entries N_{S,A_i} contains information about the mean number of occurrences of A_i in the derivation process. When the chain has been augmented, a specific nonterminal uniquely represents one or two terminals, and by keeping track of which nonterminals are associated with a specific terminal the mean number of times the terminal occurs in the strings can be computed. The sum of the mean values for all terminals gives the mean string length.

Information about the probability of terminating the derivation process, i.e., how often does the process terminate a string can be obtained by computing the stationary distribution of the chain. This can be done by changing the recurrent final state into a non-recurrent state by inserting a transition from the final state back to the starting state and then computing $\mathbf{P}(n)$ for n going toward the upper limit [46]. With this approach both the mean and the variance can be computed. Another approach would be to use an augmented chain with a special non-recurrent state indicating termination for the computation of the fundamental matrix. For more details see Appendix A.

The Markov chain representation is not capable of handling multiple entity random variables, as it does not enumerate strings explicitly, and furthermore only looks at each rewriting at a time without reference to other rewritings which have taken place previously. This also shows that characteristics related to a specific value or interval for the random variable can not be computed.

Markov chains have the capability of computing some order-dependent single entity random variables related to the derivation process and the language (only for regular languages). By computing $\mathbf{P}(n)$ it is possible to characterize the derivation process after n derivations which relate to a specific position in the strings or the derivation process.. $\mathbf{P}_{S,F}(n)$ denotes the accumulated probability mass for all the terminated strings and $P_{S,A_i}(n)$ denotes the probability that nonterminal A_i is in the sentential form at this specific point in the derivation process. Algorithm A.3 specifies the details of the computations.

5.3 The branching process model of stochastic context-free languages

If the grammar is no longer linear, the number of nonterminals in a sentential form will not be limited to one. For context-free grammars in general the number can grow and become very large at times during the derivation process but will eventually reach zero, when the derivation terminates. Such a process cannot be modelled by a Markov chain with only a finite number of states, so instead a branching process model is used [44, 45]. Such a model is capable of describing the evolution taking place, when a group of objects reproduces as itself or other objects. The evolution can be broken down in a number of generations, and the model can describe how the size of the generations evolve [26].

Let the generations be numbered 0,1,2 ... and let Z_n be the size of the nth generation, meaning the number of objects in the generation. The following assumptions are made,

- the first generation consists of exactly one object $Z_0 = 1$ indicating that the process has a unique starting situation.
- the reproduction process is governed by a probability function p for each object, where p_k is the probability that an object existing in the nth generation will have k children in the $(n+1)$th generation. It is required that $\sum_k p_k = 1$. It is assumed that the probability distribution is independent of n.
- different objects reproduce independently of one another. The number of children that a given object can produce is not dependent on e.g., the total number of objects in the generation.

If only one type of objects exists the process is called a *single-type branching process* otherwise a *multi-type branching process.*

The generating process of a stochastic context-free grammar can in general be modelled as a multi-type branching process, where the nonterminals act as the object capable of reproduction. Strings in the individual generations can consist of both nonterminals and terminals. The derivation process of a grammar relates to a multi-type branching process in the following way. The unique starting symbol of the grammar is the zeroth generation. The strings in the first generation come from rewriting the starting symbol in all possible ways according to the grammar. Strings in the second generation comes from rewriting all nonterminals in all the strings in the first generation, in all possible ways. Furthermore it contains the strings without nonterminals for the first generation. In general, the nth generation contains the new strings coming from rewritings of nonterminals in the $(n-1)$th generation and strings without nonterminals in generation 1 to $(n-1)$. An example of how the derivation process of a context-free language can be modelled by a branching process is given in Appendix B.

The derivation process of a stochastic context-free grammar with only one type of nonterminal can be modelled by a single type branching process [44, 45].

5.3.1 Converting from a grammar to a branching process

The information required for the branching process to model the derivation process of a grammar can be specified partly as generating functions and partly as matrices [26].

Two different types of generating functions are used. The first is called f^{object} and describes how an object of type *object* produces children. There exists, one such generating function for each nonterminal in the grammar. The second type is called $F_{generation}$ and there exists such one for each generation.

The generating function for an object of type i in a process having k different types is written as:

$$f^i(s_1,\ldots,s_k) = \sum_{r_0,\ldots,r_k=0}^{\infty} p^i(r_1,\ldots,r_k)s_1^{r_1}\ldots s_k^{r_k}$$

$f^i(s_1,...,s_k)$ describes the probability that one object of type i produces children of type 1 to k. $p^i(r_1,....r_k)$ is the probability that an object of type i has r_1 offspring of type 1, ..., r_k children of type k. s can be regarded as a dummy variable. The sum is over all combinations for $r_1, r_2, \ldots, r_k$.

There will be a generating function for each nonterminal in the grammar. The function for A_i is a sum of n terms, if there are n productions having A_i as the left hand side. Each production will be of the form $p_i : A_i \rightarrow \beta$, where $\beta \in (N \cup \Sigma)^\star$, and it will give rise to a term of the form $p_i s_{A_1}^{\sharp A_1} \ldots s_{A_k}^{\sharp A_k}$, where $\sharp A_i$ denotes the number of occurrences of nonterminal A_i in the right hand side of the production.

The unique starting symbol A_s of the grammar will ensure that the zeroth generation is uniquely defined as $F_0 = s_{A_s}$. The rest of the generating functions for the size of generations can be derived based on the information described so far. It is then possible to express the generating function for $n \geq 1$ as:

$$F_{n+1}(s_1,\ldots,s_k) = F_n[f^1(s_1,\ldots,s_k),\ldots,f^k(s_1,\ldots,s_k)]$$

The computation of expected values for the entire branching process can benefit from a matrix representation of the relationship between individual nonterminals and between nonterminals and terminals. A grammar with n nonterminals and m terminals will have a $n \times n$ *first moment nonterminal matrix* $\mathbf{B}$, where $b_{i,j}$ is the expected number of nonterminal A_j directly introduced, as nonterminal A_i is rewritten. For each production which will be on the form $p_i : A_i \rightarrow \beta$ the entry $b_{i,j}$ should be updated as $b_{i,j} = b_{i,j} + p_i \times \sharp A_j$, where $\sharp A_i$ denotes the number of nonterminal A_j in the right-hand side.

There also exists a $n \times m$ *first moment terminal matrix* $\mathbf{D}$, where $d_{i,j}$ represents the expected number of terminal a_j introduced, as A_i is rewritten. The entries should be updated in a way similar to the one for $\mathbf{B}$, with the only difference that it is the number of terminals which is of interest.

5.4 Characteristics based on the branching process model

The branching process model can be used as an enumerative technique to derive both single and multiple entity random variables related to terminals, nonterminals, and productions.

Algorithm B.1 in Appendix B states the details about the conversion to a branching process.

By computing F_n it is possible to describe the situation for the nth generation. By letting n approach infinity the entire language will be generated. Since the probability of a string decreases as the length increases a number m exists so that the statistics of F_m approximate that of F_∞ with an error less than some small number ϵ.

For the nonterminals the information can be extracted directly from the generating function F_n. Any term of the form $q_i s_{A_1}^{\sharp A_1} \cdots s_{A_n}^{\sharp A_n}$ expresses the information that a sentential form with $\sharp A_i$ number of occurrences of nonterminal A_i occurs with probability q_i. Multiple terms containing $s_{A_i}^{\sharp A_i}$ can exist, and the correct results will be the sum of the coefficients for each term.

To get information on terminals the $(n-1)$th generation should be computed and extra information incorporated into the generating function while going to the nth generation [45]. The extra information must relate to which terminals a nonterminal introduces. A similar strategy must be taken, if information about specific production are to be computed.

Algorithm B.2 in Appendix B states the details about computing characteristics for the nth generation.

By adding extra dummy variables to the generation function which does not become rewritten during the derivation process, it is possible to compute the accumulated information for the first n generations. The issues of interest are the same as for the nth generation. This can also be done by defining a special generating function covering the total progeny up to the nth generation [26].

The mean and the variance of any generating function can be computed using standard techniques, based on a differentiation of the generating function [45].

Branching processes can also be used to compute mean values of single entity random variables directly using the matrix formulation of the relations between the elements of the grammar without generating one string in the language [44, 45].

By computing the largest eigenvalue ρ of the first moment matrix $\mathbf{B}$ it is possible to determine if the grammar is consistent. If $\rho > 1$ then the grammar is inconsistent, meaning that the probability mass for all strings does not sum to 1. If $\rho \leq 1$ it is a consistent grammar [22].

If the grammar is consistent a nonterminal expectation matrix $\mathbf{B}^\infty$ can be computed as $\mathbf{B}^\infty = (\mathbf{I} - \mathbf{B})^{-1}$. $B_{i,j}$ denote the mean number of times A_j has been rewritten if the derivation started with A_i. The sum of elements in the row associated with the starting symbol gives the *expected derivation length.* Multiplying $\mathbf{B}^\infty$ with $\mathbf{D}$ the *expected string length* can be computed as the sum of elements in the row for the starting symbol [22]. The individual elements in the first row of $\mathbf{B}^\infty\mathbf{D}$ is the mean number of occurrences of the individual terminals. Algorithm B.3 in Appendix B states the computations in details.

Mean values of multiple entity random variables can not be computed directly since the contextual information are not perceived in the matrix formulation.

5.5 The PTL approach

This approach is somewhat different from the previous two methods. It does not explicitly model the derivation *process* of the strings. Instead it directly models the derivation sequence being the result of the derivation process. This can be achieved, if a transformation of the original grammar G into a derivation grammar DG is performed. The terminals in the derivation grammar are unique production labels which ensure that strings in the derivation language represent derivations in the original grammar.

By designing the derivation grammar in such a way that it uses the production probabilities, interpreted as a symbol, as unique production labels it becomes possible to change the domain from stochastic to non-stochastic. By the change in domain the very large amount of theoretical knowledge from formal language theory becomes applicable to this situation.

The derivation grammar is represented as a set of equations, one for each different nonterminal. From language theory methods exist to solve such systems of equations. Since the grammar and thereby the system contain information about the entire language, a solution to the set of equations represents the derivations for all strings in the language, $L(G)$.

Since the PTL-approach utilize a regular language representation of a context-free language, the solution of the equations therefore represents equivalence classes in $L(G)$. Each class consists of the set of strings having an identical commutative map.

For non-expansive grammars a closed form solution for the system of equations can be obtained. By changing the solution into a generating function, a very compact generator of derivations in $L(G)$ exists.

In the more general case including expansive grammars a closed form solution cannot be obtained, instead the solution is the complete enumerated list of derivations for strings in $L(G)$.

No matter which type of solution is obtained the stochastic information has to be re-incorporated. This is done by giving the production labels their original interpretation as numerical probabilities.

5.6 Characteristics based on the PTL-approach

The PTL-approach is an enumerative approach capable of computing both single and multiple entity order-independent random variables related to the terminals, nonterminals and the productions.

For non-expansive grammars a special generating function can be established for each of the characteristics of interest. In the general case of expansive grammars no specific generating function can be designed, instead the set of derivations for strings in $L(G)$ can be enumerated. For both expansive and non-expansive grammars the characteristics are marked by dummy variables.

Characteristic	Linear	Context-free non-expansive	Context-free expansive
Single entity			
Mean no. of a_i, A_i, p_i	Markov $\surd\surd\surd$	Markov $\div$	Markov $\div$
	Branch $\surd$	Branch $\surd\surd\surd$	Branch $\surd\surd\surd$
	PTL $\surd\surd$	PTL $\surd$	PTL $\surd$
P(specific value)	Markov $\div$	Markov $\div$	Markov $\div$
P(interval)	Branch $\surd\surd$	Branch $\surd\surd$	Branch $\surd$
	PTL $\surd\surd\surd$	PTL $\surd\surd\surd$	PTL $\surd\surd\surd$
Multiple entity			
Mean no. of a_i, A_i, p_i	Markov $\div$	Markov $\div$	Markov $\div$
	Branch $\surd$	Branch $\surd\surd$	Branch $\surd$
	PTL $\surd\surd\surd$	PTL $\surd\surd\surd$	PTL $\surd\surd\surd$
P(specific value)	Markov $\div$	Markov $\div$	Markov $\div$
P(interval)	Branch $\surd\surd$	Branch $\surd\surd$	Branch $\surd$
	PTL $\surd\surd\surd$	PTL $\surd\surd\surd$	PTL $\surd\surd\surd$

Table 5.1: Indications of the applicability of three methods for computing order-independent statistical characteristics for stochastic grammars. There are four different kinds of indications: Not possible($\div$), possible but not advisable($\surd$), acceptable solution($\surd\surd$) and recommendable solution($\surd\surd\surd$).

Any term in the generated solution of the form $q_i x^n$ denotes that the set of strings having n occurrences of random variable x has a probability q_i.

As for the branching processes it will often be advantageous to enumerate only strings until the numerical value of the characteristic has become stable within a small margin of tolerance.

For non-expansive grammars the generating functions can be used as a means of obtaining mean and variance of the random variable of interest. This is, however, not possible for the power-series representation.

5.7 An overview

From the description of the three different methods for computing characteristics it is clear that they are competing approaches in the sense that more than one approach can be used in a given situation. Table 5.1 outlines which approach can be used to compute a given characteristic.

The general impression which the table is supposed to convey is that even though the methods are competing they complement each other quite well.

For stochastic *linear* languages, including the stochastic *regular* languages, the Markov chain representation would be the best choice of model when computing mean values of single entity random variables. The Markov chain does, however, not contain any enumerative element which would allow for the registration of strings with specific characteristics, and therefore the method is not applicable for the computation of the probabilities of a random variable having a specific value or being in a specific interval. The same arguments explain why the chain is not applicable for the computation of multiple entity variables. In the situations where Markov chains cannot be used both the branching process and the PTL-approach are applicable. The PTL-approach has been recommended due to its simpler enumerative mechanism and because of its ability to form generating functions targeting directly the specific characteristic.

For *non-expansive* and *expansive* context-free languages only branching processes or the PTL-approach can be used. There is a general pattern for the two methods in both cases. Branching processes should be used when the mean values of single entity random variables are of interest. Such values can be computed using the matrix formulation of the grammar and do not involve any enumeration. The matrix formulation does, however, not allow for the computation of single entity characteristics related to a specific value or an interval for the random variable or of multiple entity mean values. Branching processes would therefore have to be used as a enumerative technique. The high level of complexity that characterizes the branching process, and which is necessary to be able to compute the characteristics for the entire language without generating a string, is not a benefit, when it comes to enumerations of a large number of generations. For *non-expansive* grammars the enumerative complexity is limited and does not impose any significant difference between the two methods, but still PTL is recommended due to its ability to form generating functions targeting directly specific characteristics. For *expansive* grammars the enumerative complexity increases, and it now becomes significant that PTL works on equivalence classes of strings having identical commutative maps and therefore has a reduced set of strings to enumerate.

	0	1	2	3	4	5	6	7
0	1	1	2	5	14	42	132	429
1	1	3	10	35	126	462	1716	6435
2	1	6	30	140	630	2772	1.20e+04	5.14e+04
3	1	10	70	420	2310	1.20e+04	6.00e+04	2.91e+05
4	1	15	140	1050	6930	4.20e+04	2.40e+05	1.31e+06
5	1	21	252	2310	1.80e+04	1.26e+05	8.16e+05	4.98e+06
6	1	28	420	4620	4.20e+04	3.36e+05	2.45e+06	1.66e+07
7	1	36	660	8580	9.00e+04	8.16e+05	6.65e+06	4.98e+07

Table 5.2: Number of different derivation trees for different applications of individual productions for a system with the equation of the form $S = aSS + bS + c$. All values in the tables expressed as exponentials are truncated values to fit the table. Therefore the values are not the exact number of trees, but they indicate the size of the count.

To get an idea of the explosion which takes place in the number of strings having identical commutative map for even a very simple grammar consult Table 5.2 which complement Table 2.1 in Chapter 2. For a grammar which can be represented as the equation $S = aSS + bS + c$ the number of different trees was computed for various combinations of uses of the productions. The table describes the number of trees for all combinations of between 0 and 7 rewritings of $S \rightarrow aSS$ and $S \rightarrow bS$. Within one row the number of rewritings of $S \rightarrow aSS$ is fixed and the number of the other production is varied.

The table clearly shows the rapid growth in the number of trees having the same commutative map. This is used as an argument for recommending the use of the PTL-approach for the computation of statistical characteristics, where enumerations are required **and** where the ordering is not important, i.e., for order-independent characteristics.

Chapter 6

An application within Syntactic Pattern Recognition

The attention is now changed from the development and description of methods for computing statistical characteristics to the use of such characteristics in pattern recognition. This chapter gives an overview of the different ways in which characteristics can be exploited in pattern recognition and specifically develops a two-stage classifier where the characteristics are used in a pre-classification. The information from the pre-classification is then exploited in the following syntactic classification in an attempt to control the recognition time and thereby possibly reduce the time. The theoretical description of the two-stage classifier will be followed by a description of an application using the classifier.

6.1 The use of statistical characteristics in pattern classification

There are various ways in which the statistical characteristics can be exploited in pattern classification. They can be grouped into two main groups:

- Methods using the characteristics in combination with a syntactic classifier.
- Methods using the characteristics for purely statistical classification.

For the latter the idea is that the structural information of the objects can be projected onto a set of statistical characteristics, which then indirectly represent the discriminative structural information. This information can then be used in a purely statistical classifier. It is possible to compute for each language the probability that strings exist with characteristics identical to those measured for the specific input. This information for each class can be exploited in a maximum likelihood classification determining the most likely class.

Another approach would be to consider each of the characteristics as a feature in a n-dimensional feature space, i.e., one feature could be the number of occurrences of terminal a_i. From the grammar it will be possible to compute the distribution of the characteristics for each class, and thereby providing sufficient information for the large set of standard statistical classifiers to be used.

For a m-class classification problem it will also be possible to define a m-dimensional feature space where $P(inputstring \mid grammar_i)$ for $i = 1, 2...m$, will be used as the m features. This approach would be of interest if discriminant information for each class was available from the relationship existing between one class and all others. As an example consider a situation where strings in a class are characterised by having characteristics very similar to one specific class **and** very dissimilar to another specific class.

For the case where the characteristics are used in combination with the syntactic classifier two situations exist depending on whether the characteristics are used before or after the syntactic classifier. The latter case relates to those situations where a syntactic classification initially is performed for each grammar, but before making the final decision numerical information in terms of the characteristics is used in addition. This follows, to some extend, the idea behind attribute grammars [48] where numerical information is processed simultaneously as the syntactic classification is performed, and the decision is taken based both on the result of the syntactic analysis and on how close the numerical values fit the numerical description of the class. In case of statistical characteristics the idea would be to use the syntactic classifier to point out the possible class and for that class to compute the probability of generating strings with characteristics similar to those of the input string. Using both types of information would make it possible to create a less detailed (and therefore faster) syntactic classifier.

The characteristics can also be used before the syntactic classifier. By measuring how well the characteristics of the input string match each of the grammars, it is possible to make some kind of sorting of the grammars, so only the most likely ones are being chosen for syntactic classification. This allows for control over the classification time and thereby has the potential for approaching the traditional problem within any classification of lowering the processing time without significant decrease in classification performance. The problem will be further specified in relation to syntactic classification in the following sections and a solution to the problem will be developed.

6.2 The problem

As stated in Chapter 1 a pattern recognition system can be in one of two different modes. In the analysis mode information about the classes is gathered and in the recognition mode this information is used in order to classify some unknown input data. The problem of interest here is related to the recognition mode.

During recognition it is often possible to devise a procedure which, when followed, will lead to a optimal classification. This typically includes measuring some kind of distance between the input string x and each of the potential classes. Consider a syntactic recognition system having n different classes. The set of classes is described by a set of grammars, $G_1, G_2, \ldots, G_n$. To achieve optimal classification a total of n syntax analyses must be performed giving $p(x \mid G_i)$ for $i = 1, 2...n$. These n probabilities can be used for maximum likelihood classification assigning x to grammar G_i if $p(x \mid G_i) = \max_{G_j}\{p(x \mid G_j)\}$. If the a priori probability $P(G_i)$ is known, the a posteriori probability $P(G_i \mid x)$ can be computed using Bayes Rule, and a classification according to Bayes Decision Rule[14] can be performed.

The optimal procedure might, however, be time consuming. The problem addressed in this chapter concerns possible improvements regarding execution time for syntactic classifiers. The problem will be approached via the following formulation:

> *Consider a recognition system with n classes. Let the time for obtaining an optimal solution be $t_{optimal}$.*
>
> *The system is asked to classify some input called DATA in a given time frame t_{real}, t_{real} is less than $t_{optimal}$.*
>
> *Which procedures can be taken to insure that the sub-optimal classification of DATA performed within t_{real}, is the best possible solution given the time-frame?*

To give an example of a situation, where the problem stated above is important, consider a dynamic vision system. Such a system could be attached to a mobile robot providing the robot with an ability to visually perceive the environment, i.e., provide the robot with information about free space or obstacles along the route while the robot is performing a transportation task. A dynamic system is one which can work in an environment which changes as time evolves, as opposed to a system requiring a fixed, never changing, environment, e.g., obstacles can move around as the robot is moving towards the goal. A dynamic vision system consists of many modules working together each performing different tasks. One of these modules will be a recognition module. To enable such a system to adapt to dynamic changes in the scene, the system must have control over its computational resources. Thereby it can focus its attention, and allocate resources toward specific subtasks. The time available for classification of one object will therefore vary constantly, and a procedure to ensure the best possible use of the resources available is required.

To specify the problem of interest it is important to note that it is the time spent in the recognition mode which is of interest and not so much the time spent in the analysis mode, while establishing the system. In other words it is considered acceptable to spent extra time in the analysis mode to prepare further information about the classes if this can reduce the recognition time.

6.3 A solution

One possible solution to the previously stated problem would be to introduce a fast pre-classification procedure by which it is possible to rank the potential classes according to how likely they are to have generated strings with statistical characteristics identical to those of the input string. The syntactic classifier could then begin with the highest ranked grammars for the syntax analysis. The size of the sub-set of classes eventually analysed will be determined by the available processing time.

The following sections will present the theoretical development of the outlined approach for the problem, and finally present experimental results providing numerical support for the approach.

6.3.1 The proposed approach

The approach uses a two-stage classifier. For each grammar the first stage, the pre-classifier, involves computing the probability by which strings exist in the language having the statistical characteristics of the input string. This is done in the recognition mode so for the pre-classifier to be fast it has to exploit information of the probability distribution for the characteristics obtained during the analysis mode.Those distributions can be computed using the methods described in Chapter 5. The output of the pre-classifier is a ranking of the grammars according to the obtained probabilities. The second stage, the syntactic classifier, can then use the ranking to control the order in which the syntax analyses are performed and thereby exploit that the most likely grammar can be tested first.

In situations in which the available processing time is very limited, only the most likely or a few of the most likely can be selected for further classification. This might give sub-optimal classification performance, but it will be the best achievable in the given time frame. The system can, however, also ensure optimal classification, if time allows for all ranked grammars to be analysed. The fact that the same approach can give both sub-optimal and optimal performance only depending on the time available is considered important. It is not required in advance to specify a fixed time reduction for the system, which otherwise would result in a sub-optimal solution even in those situations where time was not limited.

To evaluate the performance of a system based on the above approach it would be necessary to address the following issues:

1. The decrease in classification performance which necessarily will occur since it is a sub-optimal method. This also includes the sensitivity of the system to small changes in the statistics, i.e., how different are the probabilities used in the ranking for different classes.

2. The amount of time saved by using only a subset of the grammars for the syntax analysis.

The classification performance will be highly dependent on how well the statistical characteristics represent the classes and the amount of discriminant information contained in them. Two extreme situations can be identified:

1. The classes can be discriminated syntactically and there are large variations in the statistics of the characteristics.
   The statistical information will be discriminative, if the probability distributions for the characteristics vary significantly from one class to another. It would be even more discriminative, if certain primitives only appear in a limited sub-set of the classes. In either case the probabilities obtained are expected to be very different making the pre-classifier a reliable estimator of correct classification labels.

2. The classes can be discriminated syntactically, but there is only minor variations in the statistics.
   If the primitives have an almost identical distribution for each class the rank values will be quite similar and the ranking based on those values will appear almost stochastic.

It is possible to get an estimate of the discriminative power of the primitives by doing simple symbol counting and frequency estimation based on the learning-data. This can, however, only serve as a rough estimate. The correct result can only be computed based on the grammar inferred from the strings in the learning-set. Through inference the grammar will be able to describe a much larger set of strings, having a syntactic structure similar to that of the learning-strings. The statistics of the primitives will therefore not necessarily be identical for the learning-set and the language. As the recognition process is based on the language, the statistics hereof should be used.

The amount of time saved using the proposed approach is, of course, dependent on how many of the ranked grammars are selected for further classification. It is furthermore dependent on the total number of classes and the mean string length for the different classes.

Clearly the processing time spent on parsing is proportional to the number of classes. It is, however, not possible to quantify the proportionality, because for n classes only one parse is expected to take full time (the one class for which it belongs) and the remaining $n-1$ parses will only take a fraction of the parse time. The exact time used on the last group of parses is not easy to estimate, as it depends on how long each parse must proceed, before the string can be rejected. The consequence of this is that going from n to $n/2$ grammars does not necessarily mean that the processing time will be cut to half the time. It will all be depending on the timing of those taken out.

The pre-classifier should be designed so that it is fast, but still for a given application there will be a break-even point, where the time saved by lowering the number of parses performed just balance the extra time used to perform the ranking.

The parsing time for context-free languages typically has a time complexity of $O(n^2)$, so clearly the length of the string will effect the amount of processing time saved. Again it is difficult to clearly quantify the effect, since it is dependent on at which point in the string it is rejected.

It is believed that for the number of applications, where the problem stated in 6.1 is relevant, a significant subset exists having both 1) a significant number of classes and ii) discriminative information related to the occurrences of the primitives. For those cases the two-stage classifier is expected to be useful. The following sections will therefore be devoted to developing the classifier in more details.

6.3.2 System outline

Figure 6.1 gives an outline of the two-stage recognition system. If a set of characteristics can be defined, a database can be built containing information about the characteristics for each grammar. Let, as an example, a characteristic be the number of occurrences of terminal a in a string. Then the database will contain information like the probability of one a in the string, $p_a(1)$, the probability of two a's, $p_a(2)$, etc.. Establishing the database belongs to the analysis mode of the system and does not add to the recognition time.

During recognition an input string is examined to compute the characteristics for the string. Once it is computed a similarity measure between the characteristics of the string and of each of the grammars can be established. The one having characteristics most similar to the input string is considered the most likely grammar to have generated the string and is given the highest rank. The grammar having the second highest similarity measure is given the second highest rank. Every grammar is ranked following this procedure.

Based on the ranking by the pre-classifier the syntactic classification module can first perform an analysis with the highest ranked grammar, and if time permits continue with the others according to their rank.

6.3.3 Computing the rank

The efficiency of the new classification procedure depends on a fast computation of the rank. The characteristics are therefore chosen to be the number of occurrences of the individual terminals. Since the characteristics are independent of the order of symbols in the string, they can be determined by a simple counting procedure. The characteristics are furthermore independent of each other, which allows for the joint probability of the mutual existence of the characteristics in a string to be computed simply as the product of the individual probabilities.

Let the input string be any string defined on the alphabet $\Sigma = \{a_1, a_2, \cdots, a_m\}$, and let $\sharp a_i$ denote the number of terminal a_i. Then the ranking can be performed by the following 4 step procedure:

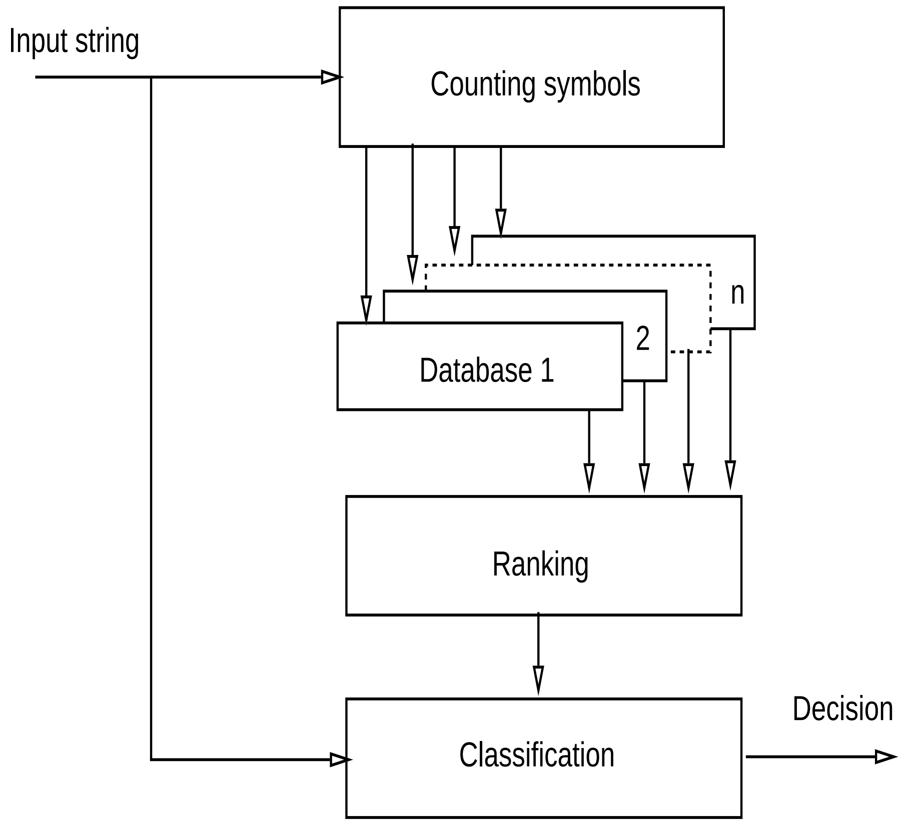


Figure 6.1: An outline of a pattern recognition system utilizing statistical characteristics during the classification.

1.

2.

3.

4.

For each input string a random variable vector V_{input} is computed.
$V_{input} = (\sharp a_1, \sharp a_2, \cdots, \sharp a_m)$.

For each of the n grammars a vector of probabilities V^n_{prob} is computed.
$V^n_{prob} = (\ \mathrm{P}(\sharp a_1), \mathrm{P}(\sharp a_2), \cdots, \mathrm{P}(\sharp a_m))$.
The element $\mathrm{P}(\sharp a_i)$ in V^n_{prob} represents the probability of the event of the random variable a_i in V_{input} given grammar n.

For each V_{input} the joint probability of grammar n generating a input with identical occurrences of primitives is computed.
$L^n_{input} = \prod_{i=1}^{c} V^n_{prob,i}$ where c denotes the number of random variables taken into account, $c = m + 1$.

The rank of grammar n given the specific input is set to L^n_{input}.

It is important to note that joint probability in those cases where the number of occurrences of one or more primitive for the string is incompatible with a given grammar can be zero. In those cases there is no need to start any syntax analysis as it is known in advance that the string will be rejected.

Computing the individual probabilities might be time consuming, and it is therefore essential that the computations are done once and then stored in a database. Thereby only a look up in the database should be executed during classification.

If a grammar G is self-embedding then $L(G)$ contains infinite many strings. For such a language an infinite number of probabilities exist for the string length and for the number of occurrences of the terminal being repeated. It is, however, a fact that the input string is finite and that very long strings have very small probabilities. It therefore seems appropriate to define a small number ϵ and discard characteristics having a probability below ϵ. This limit will be application dependent.

6.4 Experimental design

A way to test the proposed approach is implementation, and a syntactic pattern recognition system using the two-step classifier has therefore been developed. Experiments performed on the system will provide numerical support for quantifying to which extent a time reduction is possible, and the reduction in classification performance resulting from the time reduction.

The results obtained will be application dependent and the generality of the results should therefore be subjected to a critical evaluation.

6.4.1 From objects to strings

As objects 5 different types of industrial made tools; cutting nippers, flat-nose pliers, screwdriver, adjustable spanner and fixed spanner have been chosen. The tools are shown in Figure 6.2.

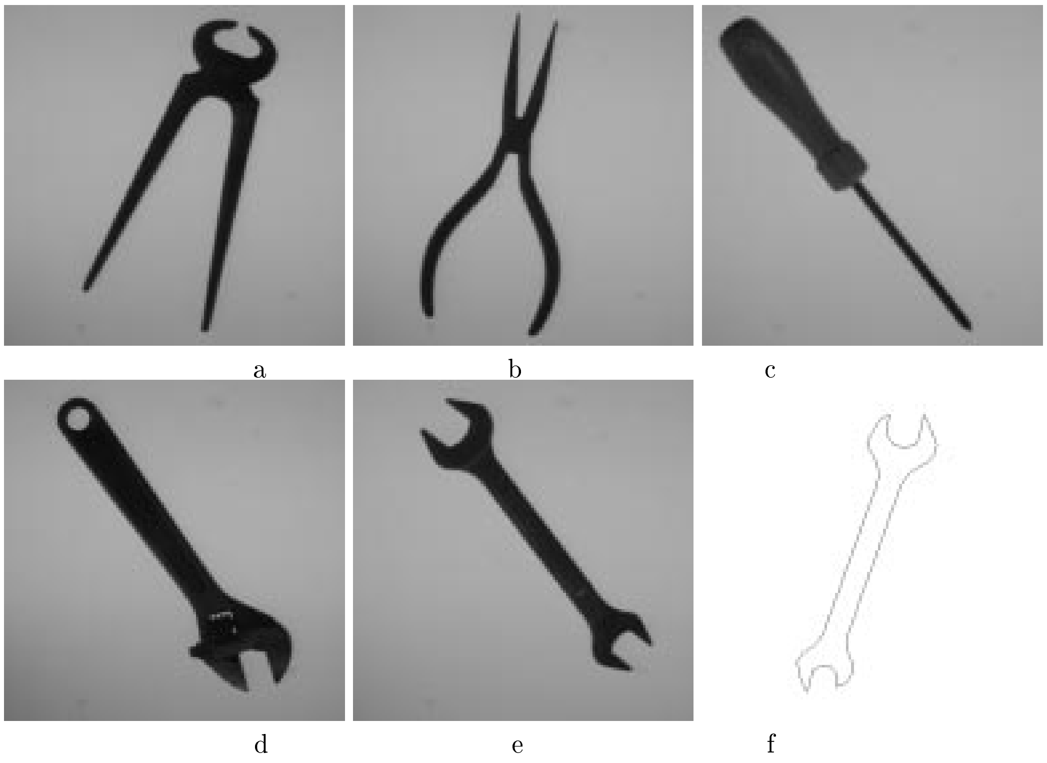


Figure 6.2: Samples of 5 tools. a) cutting nippers, b) flat-nose pliers, c) screwdriver, d) adjustable spanner, e) fixed spanner, f) boundary of fixed spanner.

For each tool a test set and a learning set each comprising 50 samples were obtained. Due to difficulties in getting a large number of different tools within each class the natural variations within a class has been simulated by varying some important parameters during data acquisition. Data has been acquired through digital images and for each image taken, the tool was relocated in the image giving different orientations. The distance to the camera has been varied to simulate objects of different size. For the tools containing adjustable parts, i.e., cutting nippers, flat-nose pliers and adjustable spanner, the adjustments were changed from image to image. Furthermore the light conditions were varied, although the general type i.e., back lighting, was used all the time.

For each image the object was segmented based on a global threshold and the contour of the segmented object was approximated by a polygon. Each object was represented by close to 70 polygon-segments (straight lines) of identical length. The length varied from object to object depending on the size of the object, but the difference in length never exceeded a factor of 2. The high number of segments was necessary to obtain sufficient resolution in the structural description. The tools had a large perimeter, but important changes in the structure took place within a small percentage of the total perimeter. The starting point of the boundary description was automatically chosen as the point on the contour farthest away from the center of mass of the object. The number of possible starting points was thereby limited. The inferred grammar

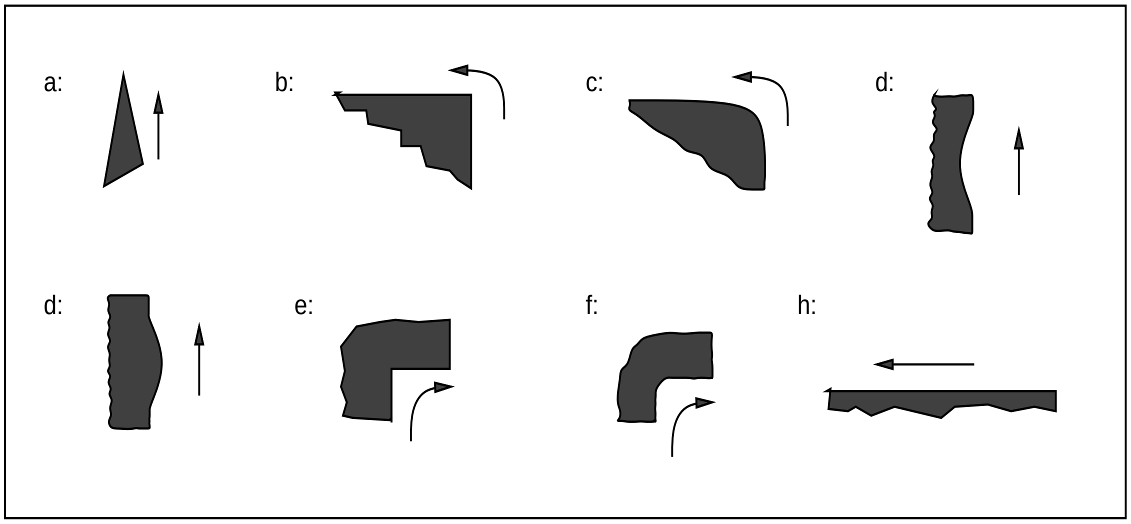


Figure 6.3: The 2D interpretation of the primitives. The shaded area represents the object, and the arrow indicates the direction in which the contour is processed.

is hereby required to handle more than one starting point.

A symbolic representation of the polygon has been obtained by assigning a letter from {A,B,C,D,E,F,G,H} to each segment denoting the angle α between the segment and its successor. Thereby the description becomes invariant to the actual orientation of the objects.

A = $337.5 < \alpha \leq 22.5$	E = $157.5 < \alpha \leq 202.5$
B = $22.5 < \alpha \leq 67.5$	F = $202.5 < \alpha \leq 247.5$
C = $67.5 < \alpha \leq 112.5$	G = $247.5 < \alpha \leq 292.5$
D = $112.5 < \alpha \leq 157.5$	H = $292.5 < \alpha \leq 337.5$

6.4.2 From symbols to characteristics

The initial symbols were, however, not suitable as primitives, since the individual symbols contain too detailed information about the contour of the object. Grammars based on such a detailed description tend to be very large. Instead the segments were collected and grouped into 7 primitives as shown in Figure 6.3. Changing from the initial symbolic description to the primitive description was realized by using the standard Unix-tool called LEX. Below, examples are given of symbolic representations before and after the primitive conversion for both cutting nippers and adjustable spanners.

Three cutting nippers in the initial representation:
```
EEEEEEEEEEEEFDEFFHDDEDCHFFFCEEEEEEEEEEEEEHEEEEEEEEEEEDDEDEEEEEEEEEEEEH
EEEEEEEEEEEEDDDEEEEEEEEEEEEEHEEEEEEEEEEEEEEDFFHDDECCHFFFDEEEEEEEEEEEEEEH
EEEEEEEEEEEEDDDEEEEEEEEEEEEHFEEEEEEEEEEEEDFFFHDCEDCHFFFDEEEEEEEEEEEEEEH
```

Same cutting nippers in their primitive representation:
hdfabbafdhahbha

hchaheabbaeha
hchaheabbaeha

Three adjustable spanners in the initial representation:
FFEEEEEEEEEEEEEEEEEEEEEEEEEEEFEEGFDDEDDEGFEEEFFDEEEEEEEEEEEEEEEEEEEEEEFFF
FFEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEGEEDEDDEGFEEEEHDDEEEEEEEEEEEEEEEEEEEEEEFFF
FFEEEEEEEEEEEEEEEEEEEEEEEEEEFEEEFHEEDDDDEGFEEEFFDEEEEEEEEEEEEEEEEEEEEEEFFF

Same adjustable spanners in their primitive representation:
fhheebbeehhf
fhhacehabhhf
fhheaceehhf

Based on the string representations of all the 50 samples of a tool in the learning set a context-free grammar was constructed by hand for each class. It was constructed in such a way that the number of strings in each of the 5 languages would be finite making completely sure that they fulfilled the requirement of being non-expansive, i.e., expansive languages will always contain infinitely many strings.

The production probabilities of the grammar were estimated based on the frequency of occurrences during a parse of all strings in the sample set[18]. The probability $p_{i,j}$ for the production $A_i \to \alpha_j$ was obtained as,

$$p_{i,j} = \frac{n_{i,j}}{\sum_j n_{i,j}}$$

where $n_{i,j}$ is the number of times production $A_i \to \alpha_j$ is used for all the samples. $\sum_j n_{i,j}$ is the number of times any production having A_j as its left-hand side was used.

As an example the stochastic grammar for the class of screwdrivers is given below.

1.00 : S $\to$ KLM	0.62 : Q $\to$ T
1.00 : K $\to$ NRN	0.38 : Q $\to$ TT
0.39 : N $\to$ eh	0.48 : T $\to$ e
0.59 : N $\to$ h	0.40 : T $\to$ d
0.02 : N $\to$ hh	0.10 : T $\to$ fd
0.88 : R $\to$ F	0.02 : T $\to$ a
0.12 : R $\to$ FF	0.50 : H $\to$ h
0.36 : F $\to$ e	0.50 : H $\to$ ha
0.59 : F $\to$ d	1.00 : L $\to$ PP
0.03 : F $\to$ de	0.78 : P $\to$ f
0.02 : F $\to$ be	0.22 : P $\to$ e
1.00 : M $\to$ HQH	

In this experiment it was chosen to use the number of occurrences of each of the 7 primitives as characteristics. Using the algorithms from Chapter 3 and implementing them in Maple V the probability function associated with the occurrences of primitives for each of the tools was computed, and is represented in Table 6.1.

By inspection of Table 6.1 it can be seen that in general the primitives are well represented in all classes. The worst situation exists for the flat-nose pliers where primitive c and f do not exist in the learning data.

Due to its generality the system was implemented based on Earley's parser.

Cutting nippers									
	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
a	0.0	0.0	0.0	0.060000	0.940000	0.0	0.0	0.0	0.0
b	0.018144	0.179568	0.506432	0.295856	0.0	0.0	0.0	0.0	0.0
c	0.440000	0.560000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
d	0.174060	0.363805	0.301890	0.127521	0.029123	0.003433	0.000164	0.0	0.0
e	0.143126	0.368583	0.336512	0.130378	0.020537	0.000861	0.0	0.0	0.0
f	0.378431	0.423314	0.168538	0.028053	0.028053	0.001662	0.0	0.0	0.0
h	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
Flat-nose pliers									
	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
a	0.0	0.0	0.002816	0.086530	0.738351	0.161094	0.010976	0.000224	0.0
b	0.015784	0.195745	0.788468	0.0	0.0	0.0	0.0	0.0	0.0
c	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
d	0.954314	0.044590	0.001093	0.0	0.0	0.0	0.0	0.0	0.0
e	0.880140	0.115513	0.004344	0.0	0.0	0.0	0.0	0.0	0.0
f	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
h	0.0	0.0	0.002058	0.000554	0.049625	0.208015	0.255009	0.238093	0.26633
Screwdriver									
	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
a	0.243138	0.493100	0.256824	0.069000	0.000038	0.0	0.0	0.0	0.0
b	0.977216	0.022304	0.000480	0.0	0.0	0.0	0.0	0.0	0.0
c	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
d	0.142449	0.419732	0.353168	0.080267	0.004382	0.0	0.0	0.0	0.0
e	0.053992	0.201695	0.312909	0.261308	0.127207	0.036488	0.005909	0.000475	0.0
f	0.041904	0.303453	0.571689	0.080639	0.002311	0.0	0.0	0.0	0.0
h	0.0	0.0	0.0	0.0	0.960400	0.039200	0.000400	0.0	0.0
Adjustable spanner									
	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
a	0.383367	0.423164	0.152152	0.033309	0.007221	0.000783	0.0	0.0	0.0
b	0.655600	0.284000	0.060000	0.0	0.0	0.0	0.0	0.0	0.0
c	0.084000	0.916000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
d	0.976000	0.024000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
e	0.003324	0.045613	0.207051	0.381906	0.267742	0.062769	0.031589	0.0	0.0
f	0.0	0.120000	0.880000	0.0	0.0	0.0	0.0	0.0	0.0
h	0.0	0.0	0.0	0.0	0.600000	0.400000	0.0	0.0	0.0
Fixed spanner									
	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
a	0.0	0.0	0.0	0.060000	0.940000	0.0	0.0	0.0	0.0
b	0.435600	0.448800	0.115600	0.0	0.0	0.0	0.0	0.0	0.0
c	0.115600	0.448800	0.435600	0.0	0.0	0.0	0.0	0.0	0.0
d	0.960000	0.040000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
e	0.053856	0.392508	0.399488	0.130302	0.021714	0.002023	0.000101	0.0	0.0
f	0.009676	0.116044	0.419804	0.413546	0.040924	0.0	0.0	0.0	0.0
h	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

Table 6.1: The probability distribution for the primitives. p_i denotes the probability of having i occurrences of a given primitive.

6.5 Results and discussion

Six different sets of tests were performed, one using all parsers without any ranking, being the optimal classifier, four tests using different numbers of ranked grammars, 4, 3, 2 and 1 respectively and one using only the rank values without performing any syntax analysis. Each test was performed for each tool and the amount of CPU time used and the classification results for the pre-classifier and the two-stage classifier were recorded.

The results for the pre-classifier are shown in Table 6.2. They show a very strong correlation between the class given first rank and the correct class. The mean classification percentage is 98, with three classes as high as 100%. The strong correlation could be the result if the primitives have been partitioned between the classes, but as shown in Table 6.1 the primitives are well represented in all classes. The correlation must therefore be due to the individual distribution of primitives within each class. This distribution must therefore have contained discriminative information.

Table 6.2 also shows that in most cases only the first and second ranked grammar could be identified. This indirectly shows that often the input string was found incompatible with a grammar. The effect of the reduced number of ranked grammars on the processing time will be discussed later. The table furthermore shows that the rank values indicate similarities between the tools which are different from the one intuitively estimated based on the structure of the objects, i.e., the cutting nippers have characteristics more similar to the screwdriver than with the flat-nose pliers. This just shows that the characteristics measure something different from the structure of the objects.

By a close inspection of the probabilities used for the ranking (not shown here) it can be seen that the distance between the probabilities is great. Usually in the range of a factor of 100 or 1000. Two probabilities were never closer than a factor 10. This is a strong indication that minor changes in the probabilities associated with the occurrences of the primitives will not effect the final ranking.

The results for the 2-stage classifier is given in Table 6.3. The timing was done using the Unix command **time**. To simplify the process of generating testresults all samples for each tool were processed in one job. The time measured is therefore the accumulated time for processing 50 strings. To reduce the influence of normal variations in the processing time an average from 10 runs was obtained for each timing result.

The results show that a time reduction can be achieved. Going from all parsers to the two most likely reduces the time by 44% and using only the most likely parser gives a 52% reduction. The classification performance was in average reduced by 0.4% and 1.2% for the two cases.

One should be very cautious when interpreting these results. As regards both the timing and the performance results it is the relative difference between the results for the three tests which is of interest more than the specific results of each test.

Tool	1. rank		2. rank		3. rank		4. rank	
	Tool	%	Tool	%	Tool	%	Tool	%
Cutting nippers	Cut. nip.	100	Screw.	54	Not def.	84	Not def.	100
			Not def.	30	Screw.	10		
			Flat.	10	Adj. span.	4		
			Fix. span	6	Fix. span.	2		
Flat-nose pliers	Flat.	100	Screw.	56	Not def.	100	Not def.	100
			Not def.	44				
Screwdriver	Screw.	100	Not def.	58	Not def.	100	Not def.	100
			Adj. span.	42				
Adjust. spanner	Adj. span.	94	Not def.	88	Not def.	100	Not def.	100
	Screw.	6	Screw.	8				
			Adj. span.	4				
Fixed spanner	Fix. span.	96	Not def	68	Not def.	86	Not def.	100
	Cut. nip	2	Cut. nip	28	Adj. span	8		
	Adj. span.	2	Adj. span.	2	Cut. nip.	4		
			Fix. span.	2	Screw.	2		

Table 6.2: Distribution of the rank values for each of the 5 tools. Not def. means that the rank-value was zero.

Tool	5 parsers		4 parsers		3 parser		2 parser		1 parser		Only rank	
	sec	%	sec	%	sec	%	sec	%	sec	%	sec	%
Cutting nippers	9.42	94	5.77	94	5.93	94	5.50	94	4.47	94	0.91	100
Flat-nose pliers	9.17	98	5.60	98	5.60	98	5.53	98	4.62	98	0.89	100
Screwdriver	9.13	98	5.20	98	5.09	98	5.20	98	4.42	98	0.94	100
Adj. spanner	9.07	88	4.62	88	4.66	86	4.67	88	4.41	86	0.96	94
Fixed spanner	9.31	100	5.15	98	4.95	98	4.94	98	4.37	96	0.99	96
Average	9.22	95.6	5.27	95.2	5.25	95.2	5.17	95.2	4.46	94.4	0.94	98

Table 6.3: Timing and classification results for the six systems using 5,4,3,2,1 parser(s) or using only the rank values. The sec parameter represents the time used to parse 50 samples measured in seconds of CPU time.

The results also show that if only the results from the pre-classifier were used a reduction in time close to 90% could be obtained. At the same time the performance surprisingly would increase by 2.4%.

The processing time is proportional to the number of syntax analyses performed. As discussed in Section 6.2.1 it can be difficult to quantify the proportionality. This is also clear from the results. Going from 5 to 1 or from 2 to 1 does not reduce the time with a factor of 5 or 2, respectively. The reason for this is that only one parser requires full time and the rest stops further processing as soon as it is detected that it can not have generated the string.

The test shows a significant drop in processing time from using no rankings at all to using the four most likely parsers. On the other hand no further decrease seems to take place reducing the number to 3 or 2 parsers. This somewhat unusual result is due to the fact that even though the system was set up to perform 4 syntax analyses, they were not all performed if the ranking did not find 4 rank values greater than zero. The timing results remain almost constant for 4, 3 and 2 parser because as shown in Table 6.2 in most cases only two grammars was chosen by the ranking.

The length of the strings in each test has been as low as 10 to 15 symbols. Since the time complexity of the parser is at least of the order $O(n^2)$ whereas the ranking is $O(n)$, the relative time spent on the ranking decreases as n increases. For the many possible applications, in which the string length exceeds 15, the time reduction would therefore exceed the one reported in this test.

6.6 Summary

This chapter has discussed various ways of exploiting statistical characteristics within pattern classification, and one has been developed, tested and discussed in details.

A two-stage syntactic pattern recognition strategy has been presented. It focuses on reducing the time spent on the syntax analysis. Optimal classification for a n-class problem requires that all n parsers are used for each input string. It has been shown that it is possible to rank a stochastic context-free grammar according to how likely it is to generate a string with the specific number of occurrences of specific terminals as observed from the input string.

A recognition system for 5 industrially made tools has been designed as a test-bed for a comparison of the optimal and this sub-optimal procedure. Going from 5 parsers to only the most likely reduces the processing time by 52% and decreases the classification by 1.2%.

The two-stage classification strategy is not claimed to be applicable to every syntactic pattern recognition system, but should be thought of as an additional tool in the large tool-box from which the system designer gets inspiration during the creation of a classification system.

Chapter 7

Summary

This chapter will summarize the work presented in the previous chapters, discuss the perspectives of the results obtained and finally touch upon some issues open for further research.

7.1 Summary of the Dissertation

As described in Chapter 1, the aim of the work presented has been:

- to determine the possibilities of using a regular language representation of a stochastic context-free language for the computation of the set of order-independent characteristics related to the strings in the language or the derivation process of this.
- to gain new insight in how such characteristics can be made useful in a syntactic pattern recognition system.

To conclude on the work described in the previous chapters the following results have been obtained:

- A *three-step Parikh-Thomason-Larsen approach* (PTL-approach) has been developed. It serves as a tool for computing order-independent statistical characteristics based on order-independent random variables associated with strings in a stochastic context-free language and the derivation process of this. The approach utilizes a regular language representation of the context-free language.
- Various ways in which statistical characteristics can be exploited in pattern classification have been outlined. To exemplify the potential a *two-stage syntactic classification strategy* has been developed and tested experimentally.

The theoretical foundation for the PTL-approach was layed out by Parikh, when he proved that the commutative map of any context-free language can be represented by a regular language. In a commutative map no information about the ordering of the strings is maintained, but since order-independent characteristics are not based on such information, Parikh theorem shows that the necessary information of the characteristics is expressible as a regular language.

The PTL-approach describes that 1) the stochastic context-free grammar should be changed into a non-stochastic context-free derivation grammar using the production probabilities as unique production labels, 2) the derivation grammar should be represented as a system of equations for which a regular language solution can be obtained, 3) the solution should be used to compute statistical characteristics of the original stochastic grammar. The approach has been supported by exact algorithms for all the detailed steps involved in the computation of characteristics.

The complexity of derivations of a stochastic context-free language is very dependent on whether the grammar is expansive or not. Therefore it was necessary to approach the two cases differently giving rise to two different implementations of step 2 and 3 in the PTL-approach.

For the *non-expansive* case it has been shown that Pilling's Method is applicable in order to obtain a solution of the system of equations representing the derivation grammar. It has furthermore been documented that the method cannot cope with the increased complexity of expansive languages. It has been shown that order-independent random variables can be computed by using a generating function representation of the solution obtained by Pilling's Method. Examples have been given showing how a generating function can be tailored to compute each characteristic of interest.

For *expansive* context-free languages, meaning context-free languages in general, attention was focussed on methods capable of handling the increased complexity of expansive grammars. In formal language theory a few examples exist where Lagrange's Inversion Formula, known from complex function theory, has been used to derive a solution of an equation representing a one-nonterminal grammar. Searching the literature a generalization of Lagrange's Inversion formula (GLIF) for complex functions of more than one variable was discovered. It has been shown in this thesis how GLIF can be used to derive a solution to a set of equations representing an expansive context-free grammar. The solution obtained by using GLIF is given in form of a power series in commutative variables and it has been shown how statistical characteristics can be obtained based on this representation of the stochastic context-free language.

Markov chains and *multi-type branching processes* can be used to model the derivation process of stochastic regular and stochastic context-free languages respectively. These models also serve as means of computing some characteristics of the languages. The thesis outlines the models and their use in relation to a derivation process. It gives an overview of the applicability of the existing methods and the PTL-approach to computations of specific characteristics.

Different ways of exploiting statistical characteristics in pattern classification have been outlined. To exemplify the potential of statistical characteristics within syntactic pattern recognition a new two-stage classification strategy has been formulated. It ensures that the best possible classification is performed even in situations, where time restrictions does not allow for the usual optimal classification strategy. The strategy uses a pre-classifier to rank the grammars according to their probability of generating strings with statistical characteristics similar to those of the input string. The syntactic classifier in stage two can then ensure that the most likely grammar is tested first. To provide numerical support for the strategy a syntactic pattern recognition system has been designed to work on five different industrially made tools. Results show that using the two-stage strategy can decrease the processing time significantly giving only a minor decrease in classification performance. These results are, however, application dependent.

7.2 Perspectives of the results

The increased possibilities for computing statistical characteristics have very naturally put forward the question of the potential applications of those characteristics. Some obvious choices can be listed:

1. to design new classification strategies where the new and additional information provided by the statistical characteristics is exploited to advance the classification performance. Various possibilities exist including the formulation of a hybrid recognition system, where structural information is integrated in a statistical framework.

2. to use characteristics during an inference procedure to ensure that the inferred grammar possess characteristics similar to those which can be extracted from the set of learning samples. This also opens up for new interesting possibilities in terms of using the characteristics to define a metric by which the difference between the learning data and the set of strings genereted by the inferred grammar can be measured.

3. to use the characteristics to guide a non-deterministic parser. Due to the non-determinism the parser must, during the processing of a string, take decisions on which rewriting to apply. Statistical characteristics related to the mean number of applications of productions, or nonterminals can possibly guide the parser towards taking the best decision.

The work presented in this thesis should be seen as one step on the road which eventually leads to a greater awareness of the significant potential of stochastic languages as models for complex patterns.

7.3 Future work

Some issues for further research can be identified.
A more in-depth analysis of the accuracy of the approximation for the random variables would be valuable for a better understanding of the practical limitations of the theoretical formulated method.

Likewise a more detailed analysis of the trade-off between the amount of computations used and the accuracy of the results obtained for Markov chains, multiple branching processes, and for the PTL-approach.

The complexity of computing characteristics for context-free languages is clearly higher than for regular languages, especially in the expansive case. It would therefore be of interest to develop a regular language *approximation* to the context-free language, which would allow characteristics to be computed and would keep the inherent error below some threshold.

Even though it is not documented in this thesis, work has been conducted in an attempt to generalize the Pilling Method to be applicable even in the expansive situation. This has so far not been successful, but it definitely still has its relevance on a list of future research areas.

Appendix A

The Markov chain model for stochastic linear languages

This appendix gives a description of how the derivation process of stochastic linear languages can be modelled by a Markov chain, and how the model can be used for computations of statistical characteristics of the language.

A.1 The Markov chain model

For a linear grammar each application of a production-rule introduces at most one new nonterminal in the sentential form. Therefore each sentential form leading to a terminated string contains exactly one nonterminal. The sequence of such nonterminals beginning with the starting symbol describes the derivation process of a string in the language. Such a sequence can be modelled by a Markov chain. Based on such a representation it is possible to compute characteristics as the status after n derivations and the mean values for the entire derivation process.

A Markov chain is a Markov process whose state space (set of possible states) is finite or countable. A Markov process is a stochastic process $\{X(t), t \geq 0\}$ where for any $t_1 < t_2 < ... < t_n$ in the index set the conditional distribution of $X(t_n)$ depends only on $X(t_{n-1})$. A Markov chain can be specified by stating the set of possible events $\{X_n\}$, some starting event X_0, and the conditional probabilities $P(X_t = i \mid X_{t-1} = j)$, being the conditional probability of observing X_j followed by X_i.

A convenient way of representing a Markov chain is by its transition diagram. It consists of states representing the events X_n and arcs labelled with the probability $p_{i,j}$ representing the probability of making a transition from state i to state j. An example is given in Figure A.1. The chain in the figure consists of 4 states $\{S, A, B, F\}$. $\{S\}$ is a starting state indicated by the small arrow and a final state $\{F\}$ indicated by the double circle. A realization of the chain is any sequence going from the starting state to the final state.

The Markov chain capable of modelling the derivation process can be characterised as a *discrete parameter, time homogeneous, final state stochastic process*

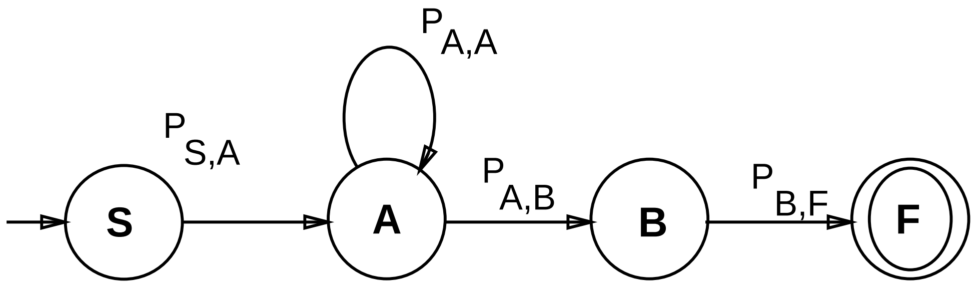


Figure A.1: An example of a Markov chain.

having the 1st order Markov property. This should become clear after the following arguments.

The derivation process of a string is clearly a stochastic process since the rewritings are governed by the production probabilities.

The rewriting of a specific nonterminal depends only on its presence in the sentential form and not on which production rule that introduced it. Decisions about the next sentential form can therefore be taken based only on the present form without knowledge of the previous forms, which is equivalent to the 1st order Markov property.

The chain represents a sequence of nonterminals coming from a finite set. Associating one state with each of the nonterminals the state space becomes finite.

Each derivation can be broken down into a set of discrete steps, one step for each rewriting. If the parameter of the chain is associated with derivation steps, the parameter becomes discrete.

Since the set of probabilistic productions and the conditions for rewritings remain the same throughout a derivation, and from one derivation to the next, the chain representing the derivation can be assumed time homogeneous.

A.2 Converting from a grammar to a Markov chain

To model the derivation process of a language $L(G)$ generated by a grammar G it is necessary to convert the grammar into a Markov chain. The information required to specify the chain is a set of states $\{X_n\}$, a probability mass function for the start configuration and a conditional probability mass function.

The probability mass function

$$p_j(n) = P[X_n = j]$$

denotes the unconditional probability of being in state j at time t_n, or being in state j after n transitions. A special situation exists for $n = 0$ where $p_j(0)$ denotes the probability of starting in state j. If there are n states $\{1, 2, 3, ..., n\}$, a n-dimensional initial probability vector $\mathbf{p}(0)$ can be constructed,

$$\mathbf{p}(0) = (p_1(0), p_2(0), \dots, p_n(0))$$

The conditional probability mass function is defined as

$$p_{j,k}(m,n) = P[X_n = k \mid X_m = j]$$

$p_{j,k}(m,n)$ gives the probability of being in state k at time t_n conditioned on being in state j to time t_m. If the chain is stationary, the notation is $p_{j,k}(n)$. For $n = 1$ it is called the *one-step transition probability* and in general the *n-step transition probability.* For notational simplicity $p_{j,k}(1)$ will be written $p_{j,k}$.

If there are n states in the chain there will be $n \times n$ one-step transition probabilities. They can be arranged in a matrix $\mathbf{P}$ so that each entry in $\mathbf{P}(1)$ is a one-step transition probability. For a Markov chain with state space {1,2,...,n} the matrix looks like

$$\mathbf{P}(1) = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & & \\ \vdots & & & \\ p_{n,1} & \cdots & \cdots & p_{n,n} \end{pmatrix}$$

The matrix is called the *transition matrix.* Note that the elements of the transition matrix have the following properties:

$$\begin{array}{ll} p_{j,k} \geq 0 & \text{for all } j,\ k \\ \sum_k p_{j,k} = 1 & \text{for all } j \end{array}$$

The required information can be extracted from the grammar using Algorithm A.1. An example of using the algorithm is given in Example A-1.

ALGORITHM A.1. Conversion to a Markov chain.

Input:

Output:

Method:

A stochastic linear grammar $G = (N, \Sigma, P, S)$, specified as
$N = \{A_1, A_2, \cdots, A_n\}$, $\Sigma = \{a_1, a_2, \cdots, a_m\}$, $S = \{A_1\}$
The production must be of the following form:
$p_i : A_i \rightarrow a_k A_j a_m$ or $p_i : A_i \rightarrow a_k$

A Markov chain represented as $\{X_n\}$, $\mathbf{p}(0)$, $\mathbf{P}(1)$.

```
1)  $\{X_n\} = N \cup \{F\}$. The new state $F$ is the final state.

2)  Initialize $\mathbf{p}(0)$ with zeros.
    Set $p_{A_1}$ to one.

3)  Initialize $\mathbf{P}(1)$ with zeros,
    For all productions do,
        If $p_i : A_i \rightarrow a_k A_j a_m$
            Set $P_{i,j} = P_{i,j} + p_i$
        If $p_i : A_i \rightarrow a_k$
            Set $P_{i,F} = p_i$
    End
```

Example A-1:

Consider a stochastic linear grammar $G = (N, \Sigma, P, S)$, $N = \{S, A, B\}$, $\Sigma = \{a, b, c, d\}$
P:

$$\begin{array}{ll} 0.7 : S \rightarrow aA & 0.4 : A \rightarrow dA \\ 0.3 : S \rightarrow bAa & 1.0 : B \rightarrow c \\ 0.6 : A \rightarrow cB & \end{array}$$

The set of states in the chain becomes:

$$\{X_n\} = \{S, A, B, F\}$$

If the elements of the vector $\mathbf{p}(0)$ are arranged in order of the states, S, A, B , F, and likewise for the rows and columns of $\mathbf{P}(1)$, the following result appears,

$$\mathbf{p}(0) = [1,0,0] \qquad \mathbf{P}(1) = \begin{pmatrix} 0 & 1.0 & 0 & 0 \\ 0 & 0.4 & 0.6 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The transition diagram for this Markov chain is the one shown in Figure A.1.

Algorithm A.1 converts all information about the derivation process of the grammar into the chain. The knowledge about the terminals is, however, not directly available from the chain. Some information is available through the nonterminals, but unless each nonterminal uniquely represents one terminal (or two in a fixed combination due to a production like $A \rightarrow a_1Ba_2$), some information will be lost through the conversion. As an example consider two productions $A \rightarrow aS$ and $B \rightarrow bS$. Information about e.g., the number of occurrences of nonterminal S is not sufficient to say anything about the occurrences of terminal a or b. It is necessary to augment the productions with new nonterminals to ensure uniqueness; $A \rightarrow aS_1$ and $B \rightarrow bS_2$. This, however, requires that all productions having S on the left hand side must be changed to include the new nonterminals. If the grammar has a production like $S \rightarrow aC$, it must be replaced by 2 new productions, $S_1 \rightarrow aC$ and $S_2 \rightarrow aC$. The same kind of problems exist for the terminating productions, e.g., $B \rightarrow d$ and $B \rightarrow e$. In the chain different states must be defined for the two productions. An algorithm for augmenting a linear grammar to ensure uniqueness in the relation between nonterminals and terminals in the grammar and afterwards converting the augmented grammar to a chain is given in Algorithm A.2. To record the relationship between the original and the augmented chain a number of sets must be computed during the augmentation process. C_{A_i} is the set containing the new states that nonterminal A_i gives rise to. C_{a_i} is a set of 2-tuples $(A_i, \sharp a_i)$ containing the states in the augmented chain associated with a_i and a count of occurrences of terminal a_i associated with A_i. C_{term} contains the new states representing terminating productions in the grammar.

Example A-2 shows how Algorithm A.2 augments a grammar and in Figure A.2 the transition diagram for the augmented chain is shown.

Example A-2:

Consider a stochastic linear grammar $G = (N, \Sigma, P, S)$, $N = \{S, A, B\}$, $\Sigma = \{a, b, c, d, e\}$,
P:

$$\begin{array}{ll} 0.7 : S \rightarrow aA & 0.4 : A \rightarrow dA \\ 0.3 : S \rightarrow bAa & 1.0 : B \rightarrow e \\ 0.6 : A \rightarrow cB & \end{array}$$

This is the grammar from Example A-1 but it is replicated for ease in comparing with the augmented grammar G' given below.

$G' = (N, \Sigma, P, S)$, $N = \{S, A_1, A_2, A_3, B, T\}$. $\Sigma = \{a, b, c, d, e\}$, P:

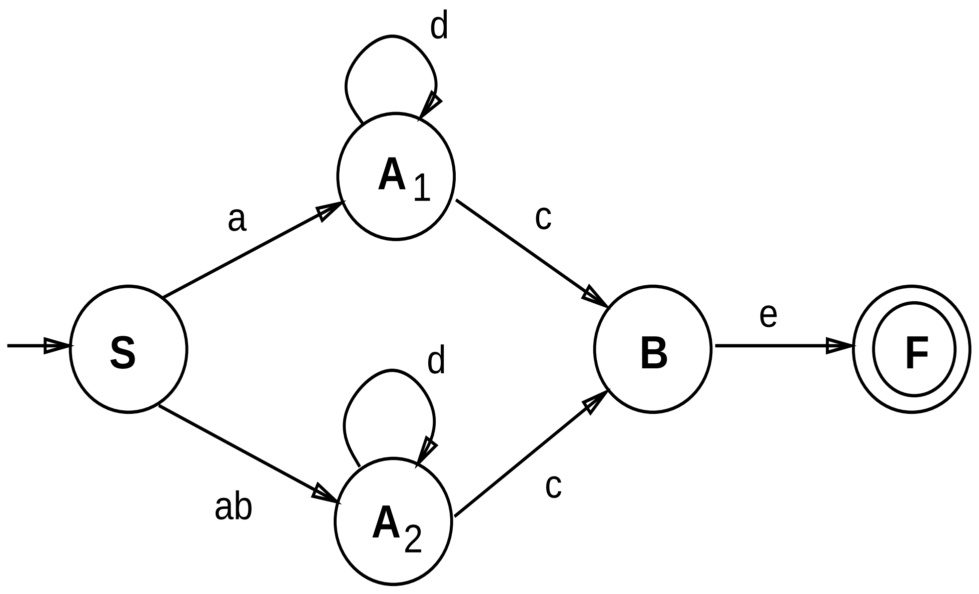


Figure A.2: The transition diagram for the augmented grammar.

$0.7 : S \to aA_1$	$0.4 : A_1 \to dA_3$
$0.3 : S \to bA_2a$	$0.4 : A_2 \to dA_3$
$0.6 : A_1 \to cB$	$0.4 : B \to eT$
$0.6 : A_2 \to cB$	$1.0 : T \to f$

The connection between the original and the augmented grammar is described in the following sets;
$C_S = \{S\}$, $C_A = \{A_1, A_2, A_3\}$, $C_B = \{B\}$.
$C_a = \{(A_1, 1)\}$, $C_b = \{(A_1, 1), (A_2, 1)\}$, $C_c = \{(B, 1)\}$, $C_d = \{(A_3, 1)\}$
$C_{term} = \{T\}$.

It should be noted that the new terminal *f* in the augmented grammar is not a valid symbol in the strings, it serves the purpose of interfacing with Algorithm A.1 for changing the result into a chain. The augmented grammar should be viewed as an intermediate result, and not as the final result of some process.

ALGORITHM A.2. Augmentation and conversion to a Markov chain.

Input:

Output:

Method:

```
A stochastic grammar $G = (N, \Sigma, P, S)$, specified as
$N = \{A_1, A_2, \cdots, A_n\}$ , $\Sigma = \{a_1, a_2, \cdots, a_m\}$, $S = \{A_1\}$
The production must be of the following form:
$p_i : A_i \rightarrow a_k A_j a_m$ or $p_i : A_i \rightarrow a_k$

A Markov chain represented as $\{X_n\}$, $\mathbf{p}(0)$, $\mathbf{P}(1)$,
$C_{A_i}$, $C_{a_i}$ and $C_{term}$.

1)  For all nonterminals $A_i$ do
        Find all productions with $A_i$ in the right hand side
        e.g., $A_j \rightarrow a_1 A_i a_2$

        Set $Diff_{A_i}$ equal to the number of different combinations
        of terminals for these productions.

        For each production do
            Substitute $A_i$ with a new unique nonterminal $A_{i,k}$
            $N = (N \setminus \{A_i\}) \cup \{A_{i,k}\}$
            $C_{A_i} = C_{A_i} \cup \{A_{i,k}\}$
            $C_{a_i} = C_{a_i} \cup \{(A_{i,k}, \sharp a_i)\}$
        End

        For all productions containing $A_i$ in the left hand side do
            Copy each production $Diff_{A_i}$ times
            Substitute $A_i$ with a new nonterminal $A_{i,k}$
        End
    End

2)  For all terminating productions do
        Set $Diff_{Terminating}$ to the number of different terminals
        in these productions.

        For each different terminal do
            Substitute $A_i \rightarrow a_i$ with $A_i \rightarrow a_i T_k$
            N = N $\cup \{T_k\}$
            Add $Diff_{Terminating}$ new productions $T_k \rightarrow f$
            $\Sigma = \Sigma \cup \{f\}$
        End
    End

3)  Apply Algorithm A.1 to the augmented grammar.
```

A.3 Computing the status after n transitions

By observing the chain after n transitions it is possible to compute the accumulated probability mass for all the strings terminated after n derivations, and the probability that nonterminal A_i is in the sentential form at this specific point in the derivation process.

All information about the status of the process after n transitions or derivation steps is stored in the unconditional probability vector

$$\mathbf{p}(n) = (p_1(n), p_2(n), ..., p_m(n))$$

which can be computed as

$$\mathbf{p}(n) = \mathbf{p}(0)\mathbf{P}(n)$$

It can be shown by using the Chapman-Kolmogorov Equation that the set of n-step transition probabilities can be determined by multiplying the one-step transition matrix with itself n times.

$$\mathbf{P}(n) = \mathbf{P}^n$$

An example of using the transition matrix and initial probability vector to describe the derivation process is given in Example A-3. This simple procedure for obtaining the status after n derivations is summarized in Algorithm A.3.

Example A-3:

A stochastic regular grammar $G = (N, \Sigma, P, S)$ is given with productions:

$$\begin{array}{l} 0.3 : S \to aS \\ 0.7 : S \to bA \\ 0.4 : A \to dS \\ 0.6 : A \to c \end{array}$$

By inspection of the grammar the transition matrix can be initiated.

$$\mathbf{P}(1) = \begin{pmatrix} 0.3 & 0.7 & 0 \\ 0.4 & 0 & 0.6 \\ 0 & 0 & 1 \end{pmatrix}$$

The matrix is formed by arranging the rows and columns in the order of the states S, A, F. The starting state is always the first and the absorbing state the last one. After some matrix computations:

$$\mathbf{P}(2) = \begin{pmatrix} 0.37 & 0.21 & 0.42 \\ 0.12 & 0.28 & 0.60 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{P}(3) = \begin{pmatrix} 0.19 & 0.26 & 0.55 \\ 0.15 & 0.08 & 0.77 \\ 0 & 0 & 1 \end{pmatrix}$$

The matrix multiplications account for all the combinations of productions in the grammar.

$p_{S,S}$ in $\mathbf{P}(2)$ represent the probability that any sentential form, after two rewritings, contains a nonterminal S , having started with S. $p_{S,F}$ represent the accumulated probability of terminated strings after two rewritings. How the values relate to the actual derivations in the grammar is shown by listening some derivations of strings.

$$
\begin{aligned}
&S \overset{0.7}{\Rightarrow} bA \overset{0.6}{\Rightarrow} bc \quad [0.42]\\
&S \overset{0.3}{\Rightarrow} aS \overset{0.7}{\Rightarrow} abA \overset{0.6}{\Rightarrow} abc \quad [0.13]\\
&S \overset{0.7}{\Rightarrow} bA \overset{0.4}{\Rightarrow} bdS \quad [0.28]\\
&S \overset{0.3}{\Rightarrow} aS \overset{0.3}{\Rightarrow} aaS \quad [0.09]
\end{aligned}
$$

The number in the square brackets is the total probability of the string, computed by multiplying the probabilities for all the rewritings used to form the string. Note that the sum of the 2 terminated strings is 0.55 which is equal to $p_{S,F}$ in $\mathbf{P}(3)$.

It can be shown that any stochastic linear grammar is consistent. Therefore the value of $P_{S,F}$ should approach 1 as n goes toward infinity. The grammar is, however, only consistent if no useless nonterminals exist in the grammar. A nonterminal A in a grammar G is useless in generating terminal strings in $L(G)$ if

1. A is not in any sentential form derivable from S, or
2. A itself does not yield any terminal string.

Useless nonterminals introduce absorbing states in the chain, in addition to the final state. These new states serve as a kind of trap state; once the process reaches the state, it remains there, unable to terminate the string. Inspection of the value of $P_{S,F}$ as n grows can detect the presence of unwanted useless nonterminals.

A.4 Computing the mean values for all derivations

Having a Markov chain representation of the grammar and using well-known properties of the chain it is possible to determine characteristics for the entire set of strings without ever generating one string. The mean string length, mean number of occurrences of terminals and nonterminals can be computed, as well as the probability of terminating the derivation process(How often does the process terminate a string?).

Information about mean values for the number of times a nonterminal or a terminal has been incorporated in a sentential form can be found by looking at the mean number of times the chain occupies the individual states. A state can be classified as either recurrent or non-recurrent. It is recurrent if the probability of ever returning to the state is one. The final state is recurrent. If

ALGORITHM A.3. Status after n derivations.

Input:

Output:

Method:

n: the number of derivations after which the status will be computed.
A stochastic grammar $G = (N, \Sigma, P, S)$, specified as
$N = \{A_1, A_2, \cdots, A_n\}$, $\Sigma = \{a_1, a_2, \cdots, a_m\}$, $S = \{A_1\}$
The production must be of the following form:
$p_i : A_i \rightarrow a_k A_j a_m$ or $p_i : A_i \rightarrow a_k$

The status represented by $P(A_i)$, $P(terminating)$.

1) Apply algorithm A.1.

2) Compute $\mathbf{P}^n$, where $\mathbf{P}$ is the output from Algorithm A.1.

3) $P(A_i) = P^n_{S,A_i}$ and $P(terminating) = P^n_{S,F}$

the probability is less than one, it is a non-recurrent state. Let N' denote the time the chain spends among the non-recurrent states until it finally is absorbed in the recurrent final state. The total time from the start until absorption is $N' = N + 1$.

Define m_j to be the *mean time to absorption* given that the chain started in state j.

$$m_j = E[N \mid X_0 = j] = E[N' \mid X_0 = j] + 1$$

m_j can be computed as the solution to the set of linear equations:

$$m_j = 1 + \sum_{k \in T} p_{j,k} m_k$$

for $j \in T$, the set of non-recurrent states. The problem is usually transformed into matrix notation. Let $\mathbf{Q}$ be the matrix

$$\mathbf{Q} = \{p_{j,k}\}$$

for j, $k \in T$. For a chain with one absorbing state F, the $\mathbf{Q}$ matrix is the transition matrix stripped for the last row and column since they are related to the recurrent absorbing state. By letting **m** denote a column vector whose probabilities are m_j the expression for the mean time to absorption becomes:

$$\mathbf{Im} = \mathbf{1} + \mathbf{Qm}$$

1 is a column vector where all the components are 1.
Rewriting the above and defining $\mathbf{N}$=$(\mathbf{I} - \mathbf{Q})^{-1}$ enables us to write an equation like

$$\mathbf{m} = \mathbf{N1}$$

N is called the *fundamental matrix* of an absorbing Markov chain. The element in the matrix $n_{i,j}$ is the mean number of occurrences of state j assuming that the process started in state i.

Since **N** is multiplied with **1**, m_j becomes a sum of all elements in row j in **N**. Since the starting symbol is unique and associated with the first row/column in the underlying transition matrix, only m_1 is of interest when using Markov chains to model the derivation process of a stochastic regular grammar.

If the grammar has been augmented, the coefficients in the fundamental matrix contain all the the information required to compute mean values of occurrences for nonterminals and terminals and the termination rate. The augmentation ensures that each state is uniquely related to a terminal and that special states are related to the terminating productions. Using the set C_A, e.g., $C_A = \{A_1, A_2\}$ the mean number of occurrences of nonterminal A in all the sentential forms is $N_{1,A_1} + N_{1,A_2}$, likewise for all nonterminals. Using as an example $C_b = \{(A_1, 2), (B, 1)\}$ the mean number of occurrences of terminal b is $(2 \times N_{1,A_1} + N_{1,B})$. Based on C_{term} the number of occurrences of a terminating state can be computed in a similar way. Based on the ratio between the number of occurrences of terminating states and the occurrence of all states, the probability of terminating a derivation P(terminating) can be computed. The mean string length can easily be computed as the sum of occurrences for each of the terminals.

The computations required to derive the characteristics are summarized in Algorithm A.4, and an example of using the algorithm is given in Example A-4.

Example A-4:

Consider a stochastic linear grammar $G = (N, \Sigma, P, S)$, $N = \{S, A, B\}$, $\Sigma = \{a, b, c, d, e\}$,
P:

$$\begin{aligned}
&0.7 : S \rightarrow aA \\
&0.3 : S \rightarrow bAa \\
&0.6 : A \rightarrow cB \\
&0.4 : A \rightarrow dA \\
&1.0 : B \rightarrow e
\end{aligned}$$

The results achieved by applying Algorithm A.2 to this grammar are given in Example A-2. The information of interest for this example is only the augmented set of nonterminals and the relation between the augmented and the original chain.

$N = \{S, A_1, A_2, A_3, B, T\}$.

$C_S = \{S\}$, $C_A = \{A_1, A_2, A_3\}$, $C_B = \{B\}$.

$C_a = \{(A_1, 1)\}$, $C_b = \{(A_1, 1), (A_2, 1)\}$, $C_c = \{(B, 1)\}$, $C_d = \{(A_3, 1)\}$, $C_e = \{(T, 1)\}$.

$C_{term} = \{T\}$.

The fundamental matrix **N** of the augmented chain is the following;

$$\mathbf{N} = \begin{pmatrix} 1.00 & 0.70 & 0.30 & 0.67 & 1.00 & 1.00 \\ 0 & 1.00 & 0 & 0.67 & 1.00 & 1.00 \\ 0 & 0 & 1.00 & 0.67 & 1.00 & 1.00 \\ 0 & 0 & 0 & 1.67 & 1.00 & 1.00 \\ 0 & 0 & 0 & 0 & 1.00 & 1.00 \\ 0 & 0 & 0 & 0 & 0 & 1.00 \end{pmatrix}$$

Using C_{A_i}, C_{a_i} and C_{term} the following characteristics can be computed.

Mean_occ(a) = 1.00	Probability(a) = 0.252
Mean_occ(b) = 0.30	Probability(b) = 0.076
Mean_occ(c) = 1.00	Probability(c) = 0.252
Mean_occ(d) = 0.67	Probability(d) = 0.168
Mean_occ(e) = 1.00	Probability(e) = 0.252

Mean_occ(S) = 1.00	Probability(S) = 0.214
Mean_occ(A) = 1.67	Probability(A) = 0.358
Mean_occ(B) = 1.00	Probability(B) = 0.214
Mean_occ(T) = 1.00	P(Terminating) = 0.214

Mean_string_length = 3.967

If the grammar has not been augmented only the mean number of occurrences for the nonterminals could have been computed by using this approach. It would however have been possible to compute the probability of terminating a derivation by looking at the stationary distribution or the long-run distribution of the Markov chain. Due to the absorbing final state the process will in the long run occupy only the final state. By changing the final recurrent state to a non-recurrent state by allowing a transition back to the starting state a non-trivial distribution can be computed. This can be done by computing $\lim_{n\to\infty} \mathbf{P}(n)$, where $\mathbf{P}(n)$ is the n-step transition matrix for the new chain without absorbing states. This technique can be used unless the chain is periodic, because then the values in $\mathbf{P}(n)$ will not converge.

A.5 Sensitivity analysis

As a hopefully illustrative example of what a stochastic grammar and characteristics computed by a Markov chain representation can be used for, the concept of sensitivity analysis for a stochastic grammar is introduced. The general idea is to measure the range of change of a specific characteristic, giving a change in the production probabilities of the grammar. This can be done using different approaches. The one taken here is to compute the sensitivity of each production based on an initial setting of the probabilities for all the productions. The approach is chosen because the result is easy to interpret. Had the sensitivity measure been conditioned on the individual settings of all the individual productions, no clear interpretation could easily be given. Different initial setting will, however, usually give different sensitivity measures. For each nonterminal A_i in the grammar it is possible to define a class $Class_{A_i}$, consisting of all the productions having A_i as the left hand side of the production rule. Within each class the sum of productions are 1. Once a production with probability, say, p_i is analysed, p_i is varied from 0.05 to 0.95. The reason for not using the full range from 0 to 1 is that applying the extreme values of such a range might introduce absorbing states in the Markov chain. It has been chosen to distribute the probability mass (1 - p_i) evenly among the rest of the productions within a class of productions.

A more formal definition is given below,

Definition 26 *The* **sensitivity of a production** $p_i : A_i \rightarrow a_i A_j a_j$ *with respect to a given characteristic C is defined as*

$$\frac{\Delta C}{\Delta p_i} = \frac{C_{0.05} - C_{0.95}}{0.9}$$

where $C_{0.05}$ and $C_{0.95}$ are the values of characteristic C computed for p_i having the value of 0.05 and 0.95 respectively.

All productions not in $Class_{a_i}$ must be kept to their initial distribution.
The probability mass (1 - P_i) should be distributed evenly among the other production within C_{A_i}.

Determining the consequences of changes in the probability distribution for at specific class of productions would be very difficult if it were based on an inspection of the grammar. To demonstrate this some results are given in Example A-5 for a grammar of moderate size.

Example A-5:

Consider a stochastic linear grammar $G = (N, \Sigma, P, S)$, $N = \{S, A, B, C, D\}$, $\Sigma = \{a, b, c, e\}$, P:

$0.4 : S \rightarrow aB$	$0.6 : B \rightarrow cAc$
$0.4 : S \rightarrow aBa$	$0.4 : B \rightarrow cD$
$0.2 : S \rightarrow cS$	$0.3 : C \rightarrow eS$
$0.3 : A \rightarrow bAc$	$0.7 : C \rightarrow bCa$
$0.2 : A \rightarrow cB$	$0.8 : D \rightarrow aB$
$0.5 : A \rightarrow bCa$	$0.2 : D \rightarrow b$

Computing the sensitivity for the mean string length gives the following results, where Sen(production) denotes the sensitivity of the specified production;

Sen($S \rightarrow aB$):	1.081491	Sen($B \rightarrow cAc$):	48.736336
Sen($S \rightarrow aBa$):	0.000008	Sen($B \rightarrow cD$):	48.735985
Sen($S \rightarrow cS$):	6.278671	Sen($C \rightarrow eS$):	10.581993
Sen($A \rightarrow bAc$):	14.814831	Sen($C \rightarrow bCa$):	10.582031
Sen($A \rightarrow cB$):	6.645400	Sen($D \rightarrow aB$):	16.596167
Sen($A \rightarrow bCa$):	4.093569	Sen($D \rightarrow b$):	16.596138

The results show significant differences in the sensitivity from one production class to another. For classes with more than 2 productions, large differences can be observed within a class. A detailed study of the structure of the grammar might have given an indication of some of the differences, but some numerical analysis like this sensitivity analysis is required if the differences should be quantified.

ALGORITHM A.4. Mean values for all derivations.

Input:

Output:

Method:

A stochastic grammar $G = (N, \Sigma, P, S)$, specified as
$N = \{A_1, A_2, \cdots, A_n\}$, $\Sigma = \{a_1, a_2, \cdots, a_m\}$, $S = \{A_1\}$
The production must be of the following form:
$p_i : A_i \to a_k A_j a_m$ or $p_i : A_i \to a_k$

Mean_occ(a_i), Mean_occ(A_i), Mean_string_length, P(terminating)

1) Apply Algorithm A.2.

2) Create $\mathbf{Q}$ as the transition matrix stripped for the entries relating to the recurrent final state.

3) Compute the fundamental matrix $\mathbf{N}$ as $\mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1}$.

4) Compute the mean number of occurrences of terminal a_i as
Mean_occ(a_i) $= \sum_{A_i} N_{1,A_i} \times \sharp a_i$,
where $\sharp$ comes from $C_{a_i} = \{A_i, a_i\}$.

5) Mean_string_length $= \sum_{a_i} Mean\_occ(a_i)$.

6) Mean_occ(A_i) $= \sum_{k \in C_{A_i}} N_{1,A_k}$.

7) Compute the probability of terminating a derivation as:
Mean_occ(terminating) $= \sum_{A_k \in C_{term}} N_{1,A_k}$.
Mean_occ(all states) $= \sum_{all\,states} N_{1,A_k}$.
P(terminating) = Mean_occ(terminating)/Mean_occ(all states).

Appendix B

The branching process model for stochastic context-free languages

This appendix describes how the derivation process of any stochastic context-free language can be modelled by a multi-type branching process. It shows how the model can be used for computations of statistical characteristics of the language.

B.1 The branching process model

If the grammar is no longer linear, the number of nonterminals in a sentential form will not be limited to one. For context free grammars in general the number can grow and become very large at times during the derivation process but will eventually reach zero when the derivation terminates. To model such a derivation process by a Markov chain would require one state for each possible configuration of nonterminals in the sentential form. In general the number can be infinite and the model becomes an infinite state Markov chain. Such a model is considered of only very little practical interest for the problem addresses there.

Instead of modelling the sequence of the nonterminals in the sentential form the focus is changed toward a more general model capable of describing the evolution taking place when a group of objects reproduces as itself or other objects. The evolution can be broken down in a number of generations, and the model can describe how the size of the generations evolves. Such a model is termed a branching process model.

Let the generations be numbered 0,1,2 ... and let Z_n be the size of the nth generation, meaning the number of objects in the generation. A branching process can be described as a Markov chain $\{Z_n; n = 0, 1, 2..\}$ for the size of the individual generations. The following assumptions are made,

- the first generation consists of exactly one object $Z_0 = 1$ indicating that the process has a unique starting situation.

- the reproduction process is governed by a probability function p for each object, where p_k is the probability that an object existing in the nth generation will have k children in the $(n+1)$th generation. It is required that $\sum_k p_k = 1$. It is assumed that the probability distribution is independent n.

- different objects reproduce independently of each other. The number of children that a given object can produce is not dependent on e.g., the total number of objects in the generation.

If only one type of objects exists the process is called a *single-type branching process* otherwise a *multi-type branching process.*

The generation process of a stochastic context-free grammar can in general be modelled as a multi-type branching process, where the nonterminals acts as the object capable of reproduction. The derivation process of a grammar relates to a multi-type branching Process in the following way. The unique starting symbol of the grammar is the zeroth generation. The strings in the first generation come from rewriting the starting symbol in all possible ways according to the grammar. Strings in the second generation comes from rewriting all nonterminals in all the strings in the first generation in all possible ways. Furthermore it contains the strings without nonterminals for the first generation. In general, the nth generation contains the new strings coming from rewritings of nonterminals in the $(n-1)$th generation and strings without nonterminals in generation 1 to $(n-1)$. This is illustrated in Example B-1.

Based on the following arguments it can be shown that the derivation process of a stochastic context free grammar fulfills the formal requirements of a branching process.

A grammar has a unique starting nonterminal which can play the role of the zeroth generation.

Under the assumption that the grammar is proper, the rewritings of nonterminals are governed by a probability function.

A rewriting of a nonterminal is only dependent on the existence of that nonterminal in the sentential form, and not on any other nonterminal which might exist in the sentential form. Therefore rewritings are independent of each other and different occurrences of the same type.

The size of a generation is the number of nonterminals in all of the sentential forms present in that generation. Any change in the number of nonterminals in a sentential form is related only to a difference in the number of nonterminals in the left and the right hand side of the production used to create the new sentential form. The use of a production is related only to the occurrence of nonterminal in the form and not on how it were introduced into the form. Therefore the size of the sentential forms makes up a 1st order Markov process. It is furthermore a chain since the size of a generation can be described by a set of discrete values, the set of non-negative integers.

From the branching process model at least three types of information can be obtained.

- The status for the nth generation.
- Accumulated information for generation $0, 1, 2, ..., n$.
- Information about the entire process.

The information for the entire process can immediately be interpreted in terms of the entire derivation process for all strings in the language, but otherwise caution should be shown in the interpretation since there is no one-to-one correspondence between elements in the nth generation and the set of sentential forms after n rewritings. The following section describes how the information about the grammar is converted into a branching process, and how the above-mentioned information can be computed.

B.2 Converting from a grammar to a branching process

The information required for the branching process to model the derivation process of a grammar can be specified partly as generating functions and partly as matrices.

Two different types of generating functions are used. The first is called f^{object} and describes how an object of type *object* produces children. There exists such a generating function for each nonterminal in the grammar. The second type is called $F_{generation}$, and there exists such one for each generation.

The generating function for an object of type i in a process having k different types is written as:

$$f^i(s_1, \ldots, s_k) = \sum_{r_0,\ldots,r_k=0}^{\infty} p^i(r_1, \ldots, r_k) s_1^{r_1} \ldots s_k^{r_k}$$

$f^i(s_1, ..., s_k)$ describes the probability that one object of type i produces children of type 1 to k. $p^i(r_1, ...r_k)$ is the probability that an object of type i has r_1 offspring of type 1, ..., r_k children of type k. s can be regarded as a dummy variable. The sum is over all combinations for $r_1, r_2, \ldots, r_k$. Theoretically the number of combinations can be very large but in any practical situation modelling stochastic grammars, the limited number of nonterminals in a grammar and the limited number of nonterminals in the right hand side of a production will ensure a limited number of combinations. The generating function is created by first forming equivalence classes, C_{A_i}, on the set of productions. Class C_{A_i} contains all the productions having A_i as the left hand side. Each class gives rise to a generating function. It consists of a sum of elements, each carrying the information from one production about the production probability and the number of occurrences of the individual nonterminals. Information about the terminals not represented in the generating function.

The unique starting symbol A_s of the grammar will ensure that the zeroth generation is uniquely defined as $F_0 = s_{A_s}$. The rest of the generating functions for the size of generations can be derived based on the information described so far, using one of the fundamental theorems of branching processes [26] which state;

$$F_{n+N}(s_1, \ldots, s_k) = F_n[F_N(s_1, \ldots, s_k)]$$

which implies that the generating function for one generation can be computed as iterates of the previous generations generating functions. It is then possible to express the generating function for $n \geq 1$ as:

$$F_{n+1}(s_1, \ldots, s_k) = F_n[f^1(s_1, \ldots, s_k), \ldots, f^k(s_1, \ldots, s_k)]$$

The computation of expected values for the entire branching process can benefit from a matrix representation of the relationship between individual nonterminals and between nonterminals and terminals. A grammar with n nonterminals and m terminals will have a $n \times n$ *first moment nonterminal matrix* $\mathbf{B}$, where $b_{i,j}$ is the expected number of nonterminal A_j directly introduced as nonterminal A_i is rewritten. There also exist a *first moment terminal matrix* $\mathbf{D}$, where $d_{i,j}$ represents the expected number of terminal a_j introduced, as A_i is rewritten. Each row in the 2 matrices contains information from a specific equivalence class for the productions.

A detailed description for creating the generating functions and the first moment matrices is given in Algorithm B.1, and some results obtained using the algorithm are given in Example B-1.

Example B-1

Consider a grammar $G = \{N, \Sigma, P, A\}$ with productions

$$\begin{array}{ll} p_1 : A \to aAB & p_4 : B \to bBB \\ p_2 : A \to d & p_5 : B \to aA \\ p_3 : B \to c & \end{array}$$

The grammar contains 2 classes of productions. For each of those a generating function is created using Algorithm B.1.

$$f_A(s_A, s_B) = p_1 s_A s_B + p_2$$
$$f_B(s_A, s_B) = p_4 s_B^2 + p_5 s_A + p_3$$

Based on the generating functions it is possible iteratively to compute the generating functions for the individual generations.

$$\begin{array}{r} F_0 = \\ F_1 = f^A(s_A, s_B) = \\ F_2 = F_1[f^A(s_A, s_B), f^B(s_A, s_B)] = \\ = \end{array}$$

$$\begin{array}{l} s_A \\ p_1 s_A s_B + p_2 \\ p_1(p_1 s_A s_B + p_2)(p_4 s_B^2 + p_5 s_A + p_3) + p_2 \\ p_1^2 p_4 s_A s_B^3 + p_1^2 p_5 s_A^2 s_B + p_1 p_2 s_B^2 + \\ p_1^2 p_3 s_A s_B + p_1 p_2 p_5 s_A + p_1 p_2 p_3 + p_2 \end{array}$$

The relationship between the content of the generating function and strings in the language becomes clear when observing some sentential forms of the language. Below is listed the forms accounted for in the second generation.

$A \Rightarrow d$
$A \Rightarrow aAB \Rightarrow adB \Rightarrow adc$
$A \Rightarrow aAB \Rightarrow adB \Rightarrow adbBB$
$A \Rightarrow aAB \Rightarrow adB \Rightarrow adaA$
$A \Rightarrow aAB \Rightarrow aaABB \Rightarrow aaABc$
$A \Rightarrow aAB \Rightarrow aaABB \Rightarrow aaABbBB$
$A \Rightarrow aAB \Rightarrow aaABB \Rightarrow aaABaA$

Going from one generation to the next <u>all</u> nonterminals are rewritten simultaneously. For linear grammars only the content of the nth generation is sentential forms after n individual rewritings. In general more than n rewritings are required to produce sentential forms in the nth generation.

ALGORITHM B.1. Conversion to a branching process.

Input:

Output:

Method:

A stochastic context free grammar $G = (N, \Sigma, P, S)$,
specified as $N = \{A_1, A_2, \cdots, A_n\}$, $\Sigma = \{a_1, a_2, \cdots, a_m\}$, $S = \{A_1\}$
The production must be of the following form:
$p_i : A_i \to \beta$, where $\beta \in (N \cup \Sigma)^\star$

A branching process represented as a set of generating functions $\{f^{A_1}, f^{A_2}, ..., f^{A_n}\}$ and the nonterminal and the terminal first moments matrices **B** and **D**.

```
1)    / * Computes the generating functions */
      For all productions $P$ do
          if $P = p_i : A_k \to \beta$ then
              $C_{A_k} = C_{A_k} \cup \{p_i : A_k \to \beta\}$
          End
      End

      For each $C_{A_i}$ do
          For each production $p_i : A_i \to \beta$ do
              $f^{A_i}_{part} = p_i s^{\sharp A_1}_{A_1} s^{\sharp A_2}_{A_2} \cdots s^{\sharp A_k}_{A_k}$,
              where $\sharp A_1$ denotes the number of occurrences
              of $A_1$ in the right-hand side, etc..
          $f^{A_i} = f^{A_i} \cup f^{A_i}_{part}$
          End
      End

2)    /* Computing the matrices */
      For each $C_{A_i}$ do
          For each production $p_i : A_i \to \beta$ do
              $b_{i,j} = b_{i,j} + p_i \times \sharp A_j$,
             where $\sharp A_j$ denotes the number of nonterminals $A_j$
             in the right-hand side.
              $d_{i,j} = d_{i,j} + p_i \times \sharp term_j$,
              where $\sharp term_j$ denotes the number of occurrences of
              terminals $term_j$ in the right-hand side.
          End
      End
```

From the previous listing string d is represented in the process as p_2, the probability of using the production rewriting A as d. Likewise string adc is represented as $p_1 p_2 p_3$. The sentential form $aaABbBB$ is represented as $p_1^2 p_4 s_A s_B^3$, because 3 productions have been used to create the form, p_1 twice, p_4 once, and the form contains one A and three occurrences of B. The first moment matrices can be found to have the following content,

$$\mathbf{B} = \begin{pmatrix} p_1 & p_1 \\ p_5 & 2 \times p_4 \end{pmatrix}$$

$$\mathbf{D} = \begin{pmatrix} p_1 & 0 & 0 & p_2 \\ p_5 & p_4 & p_3 & 0 \end{pmatrix}$$

For $\mathbf{B}$ the rows and columns are ordered as A, B. For $\mathbf{D}$ the rows are ordered as A, B and the columns as a, b, c, d. The coefficient $b_{B,B}$ is 2 times p_4 since a production with probability p_4 introduces two occurrences of B.

B.3 The status for the nth generation

For both nonterminals and terminals it is possible to compute statistical information such as the probability of a specific symbol occurring a specific number of times, the mean and the variance for the occurrence of a symbol, but it requires different techniques.

For the nonterminals the information can be extracted directly from the generating function F_n. Any term of the form $q_i s_{A_1}^{\sharp A_1} \cdots s_{A_n}^{\sharp A_n}$ expresses the information that a sentential form with $\sharp A_i$ number of occurrences of nonterminal A_i occurs with probability q_i. Multiple terms containing $s_{A_i}^{\sharp A_i}$ can exist and the correct results will be the sum of the coefficients for each term. The mean and the variance can be computed using a well-known property for generating functions[26]; for a random variable RV with generating function $F(s)$, the mean is the first derivative $F'(s)$ evaluated at $s = 1$,

$$E(RV) = F'(s) \mid_{s=1} = F'(1)$$

and the variance is computed as,

$$Var(RV) = F''(1) + F'(1) - F'(1)^2$$

To incorporate information about the terminals an additional set of generating functions must be created. There should be one function for each nonterminal, and it must contain the probability of occurrence of a specific terminal and the number of occurrences given that the nonterminal was rewritten in all possible ways. The basic idea is then to iterate the generating function for the size of the generation up to the $(n-1)$th generation and then to take the last step into the nth generation using the new additional generating functions. If we let $f^{(A_i,a_i)}$ denote the generating function for occurrences of terminal a_i introduced while rewriting nonterminal A_i, and let F_{n,a_i} denote the function for the number of terminal a_i introduced in the nth generation then it can be computed as,

$$F_{n,a_i} = F_{n-1}(s_{A_1}, \cdots, s_{A_n}) \mid_{s_{A_i}=f^{(A_i,a_i)}}$$

The method used for incorporating information about the terminals can also be used to incorporate information about the production, the only difference is the property that is associated with the dummy variable s.

The methods available to compute statistical information for the nth generation is described in details in Algorithm B.2, and an example of results obtained applying the algorithm to the grammar from the previous example is given in example B-2.

Example B-2

Given the grammar from Example B-1 it might be interesting to compute, $Prob_2(B, 1)$, the probability of sentential forms in the second generation having one occurrence of nonterminal B. In the previous example we found that

$$F_2(s) = p_1^2 p_4 s_A s_B^3 + p_1^2 p_5 s_A s_B + p_1 p_2 s_B^2 + p_1^2 p_3 s_A s_B + p_1 p_2 p_5 s_A + p_1 p_2 p_3 + p_2$$

By summing the coefficients of the terms including one occurrence of s_B we get

$$Prob_2(B, 1) = p_1^2 p_3 + p_1^2 p_5$$

Clearly any type of occurrence of symbols can be computed in a similar way e.g., the probability of having at least one occurrence of B, or the joint probability of having one A and one B, and so on. The mean value for occurrences of B in the second generation can be computed as

$$Mean(B) = F_2'(s) \mid_{s=1} = 3p_1^2 p_4 + p_1^2 p_5 + 2p_1 p_2 + p_1^2 p_3 + p_1 p_2 p_5$$

The variance can be computed as,

$$Var(B) = F_2''(s) \mid_{s=1} + Mean(B) - Mean(B)^2 =$$

$$6p_1^2 p_4 + 2p_1^2 p_2 + 3p_1^2 p_4 + p_1^2 p_5 + 2p_1 p_2 + p_1^2 p_3 + p_1 p_2 p_5 - Mean(B)^2$$

If it is of interest to compute probabilities for e.g., terminal a in the second generation, 2 new generating functions must be established. $f^{A,a}(s_a) = p_1 s_a$ for introducing terminal a from nonterminal A and $f^{B,a} = p_1 s_a$ for nonterminal B.
If these are inserted in $F_1(f^A, f^B)$, we get the following results

$$F_1(s_a) = p_1 p_1 s_a p_5 s_a = p_1^2 p_5 s_a^2 + p_2$$

from which we can compute

$$\begin{aligned} Prob(a,2) &= \\ Mean(a) &= \\ Var(a) &= \\ &= \end{aligned}$$

$$\begin{array}{l} p_1^2p_5 \\ 2p_1^2p_5 \\ 2p_1^2p_5 + 2p_1^2p_5 - 4p_1^2p_5^2 \\ 4p_1^2p_5 - 4p_1^2p_5^2 \end{array}$$

A similar approach can be taken to compute information on the use of the productions.

ALGORITHM B.2. Information for the nth generation.

```
                              Input:
                              Output:
                              Method:
A stochastic context free grammar $G = (N, \Sigma, P, S)$,
The symbols of interest and the number of occurrences $(A_i, \sharp A_i)$, $(a_i, \sharp a_i)$.

Prob$(A_i, \sharp A_i)$, Mean$(A_i)$, Var$(A_i)$
Prob$(A_i, \sharp a_i)$, Mean$(a_i)$, Var$(a_i)$

1)  / * Information about nonterminals */
    Apply Algorithm B.1 to get $f^{A_1}, \cdots, f^{A_n}$.
    Set $F_1 = f^{A_1}$, assuming that $A_1$ is the starting symbol.
    Iterate $n$ times;
        $F_n = F_{n-1}(f^{A_1}, \cdots, f^{A_n})$
    Represent $F_n$ as $F_n = \alpha_0 + \alpha_1 s_{A_i}^{\sharp A_i} + \cdots + \alpha_k s_{A_i}^{\sharp A_i}$
    where $\alpha_i \in (p_i \cup (N \setminus A_i))$

    Prob$(A_i, \sharp A_i) = \sum_{k=1} \alpha_k \mid_{s_{A_j}=1, A_j \neq A_i}$

    Mean$(A_i) = \frac{\partial F_n(f^{A_1}, \cdots, f^{A_n})}{\partial A_i} \mid_{A_1, \cdots, A_n = 1}$

    Var$(A_i)$= $\frac{\partial^2 F_n(f^{A_1}, \cdots, f^{A_n})}{\partial A_i} \mid_{A_1, \cdots, A_n = 1} + Mean(A_i) - Mean(A_i)^2$

2)  / * Information about terminals */
    Compute the equivalence classes $C_{A_i}$ for the productions.
    **For** each $C_{A_i}$ **do**
        **For** each production $p_i : A_i \to \beta$ **do**
            $f_{part}^{(A_i, a_i)} = p_i s_{a_i}^{\sharp a_i}$,
            where $\sharp a_i$ denotes the number of occurrences
            of $a_1$ in the righthandside.
        $f^{(A_i, a_i)} = f^{(A_i, a_i)} + f_{part}^{(A_i, a_i)}$
        **End**
    **End**

    Iterate $n-1$ times to create $F_{n-1}$
    Define $F_{n,a_i}$ to be the generating function for the number of ter-
    minal $a_i$ introduced in the $n$th generation,
        $F_{n,a_i} = F_{n-1}(s^{A_1}, \cdots, s^{A_n}) \mid_{s^{A_i} = f^{(A_i, a_i)}}$

    Prob$(a_i, \sharp a_i)$, Mean$(a_i)$, Var$(a_i)$ can be computed following the
    principles given in 1)
```

B.4 Accumulated information for the first n generations

In situations where it is of interest to accumulate information as the process develops, it is necessary to attach new dummy variables to the existing generating functions to capture the property of interest. The reason for this is partly that the dummy variable for the nonterminals, s_{A_i} does not contain info about properties occurring when the nonterminal is rewritten, and partly that the variable disappears when the nonterminal is rewritten. A terminated string is represented in the process as a coefficient being the probability of the string, but no knowledge about the string is attached to the coefficient.

Augmenting the generating function from example 6 to contain accumulated information about e.g., the terminal a, it will take the following form, where z is the new dummy variable,

$$f_{A,a}(s_A, s_B) = p_1 s_A s_B z + p_2$$

$$f_{B,a}(s_A, s_B) = p_4 s_B^2 + p_5 s_A z + p_3$$

The generating functions $F_0, \cdots, F_2$ would then look like,

$$\begin{aligned} F_0 &= \\ F_1 = f^A(s_A, s_B) &= \\ F_2 = F_1[f^A(s_A, s_B), f^B(s_A, s_B)] &= \\ &= \end{aligned}$$

$$\begin{array}{l} s_A \\ p_1 s_A s_B z + p_2 \\ p_1(p_1 s_A s_B z + p_2)(p_4 s_B^2 + p_5 s_A + p_3) z + p_2 \\ p_1^2 p_4 s_A s_B^3 z^2 + p_1^2 p_5 s_A^2 s_B z^3 + p_1^2 p_3 s_A s_B z^2 + \\ p_1 p_2 p_4 s_B^2 z + p_1 p_2 p_5 s_A z^2 + p_1 p_2 p_3 z + p_2 \end{array}$$

From this function it is possible to compute the probability of terminal a occurring a specific number of times in terminated and nonterminated strings <u>up to</u> the nth generation. By setting $s_1, \cdots, s_n = 0$ the probability for occurrences of a's in only the terminated strings can be computed. Clearly the mean and the variance can be computed using z as the variable for the differentiation.

B.5 Information about the entire derivation process

The most important information about the entire process is whether or not the process eventually will terminate which can be formulated in terms of a probability of extension. If the probability of extension is one, the process will terminate, if it is less than one, there will be a non-zero probability that the generation process continues forever.

In terms of grammars the problem is about finite or infinite strings, and about being consistent or not. Recall from Chapter 2 that a grammar is consistent if $\sum_{x \in L} p(x) = 1$. If a process has a probability of extension less than one, it means that the derivation process always contain sentential forms with non-terminals which can be rewritten, and therefore no terminated string can be produced. If so, the sum of probabilities for the set of terminated strings can never fully reach one which is necessary for the grammar to be consistent. Consider a grammar having only the 2 following productions, $p_1 : S \to aS$ and $p_2 : S \to b$. The non-stochastic characteristic, grammar for this grammar will generate an infinite number of strings. For the stochastic version of this grammar the situation is different, only for the trivial case when $p_1 = 1, p_2 = 0$, the number of strings will be infinite, otherwise the process is guaranteed to stop, because eventually $p_2 : S \to b$ will be used to terminate a string. The situation described here is that of a linear grammar and it can be shown in general[22] that stochastic linear grammars are consistent.

Consider a non-linear grammar having the following 2 productions, $p_1 : S \to aSS$ and $p_2 : S \to b$. Only for specific values of p_1 the grammar will be consistent. Since the grammar has only one nonterminal, it can be modelled by a single type branching process. For those it is very easy to determine consistency. It can be shown [22] that consistency occurs if $F'(1) \leq 1$. In this situation we get the following results,

$$f^S(s_S) = p_1 s^2 + p_2$$

$$F'(1) = 2p_1 < 1 \Rightarrow p_1 < 1/2$$

As long as p_1 lies in the range from 0 to 0.5, the process will be guaranteed to terminate.

In the more general case of multi-type branching processes consistency can be determined by evaluation of the largest eigenvalue of the first moment nonterminal matrix. Let $\mathbf{q}$ be the vector of extension probabilities. q^i is the probability of extension, given that the process starts with an object of type i. If all the first moments are nonnegative and finite, and $\mathbf{B}$ is positively regular (There exist a positive integer k such that B^k has only non-zero values), then $\mathbf{B}$ will have a characteristic eigenvalue ρ, which has a greater absolute value than any other eigenvalue. It can be shown [26] that if $\rho \leq 1$ then $\mathbf{q} = (1,1,\ldots,1)$. If $\rho > 1$ then $(0,0,\ldots,0) \leq \mathbf{q} < (1,1,\ldots,1)$ and $\mathbf{q}$ satisfies the equation:

$$\mathbf{q} = \mathbf{f}(\mathbf{q})$$

If $\rho > 1$, $\mathbf{q}$ can be found by simultaneously solving a system of equations. The value of q^i for i being the starting symbol can be interpreted as a relative frequency of the long-term rate of derived terminal strings.

When using multi-type branching processes to model the derivation process of a stochastic context-free language the values of $\mathbf{B}$ will always be nonnegative and finite because they are products of production probabilities and number of occurrences of nonterminals. Each individual grammar has to be checked to see if it is positively regular.

Example B-3.

Consider a grammar $G = (N, \Sigma, P, S)$ with productions:

$0.60 : S \rightarrow aSbS$	$0.45 : A \rightarrow aSbBb$	$0.6 : B \rightarrow BabB$
$0.40 : S \rightarrow bAb$	$0.50 : A \rightarrow AbAb$	$0.40 : B \rightarrow abA$
$0.05 : A \rightarrow aba$		

By using Algorithm B-1 and B-3 to determine the first moment nonterminal matrix and to compute the eigenvalues we get the result:

$$\mathbf{B} = \begin{pmatrix} 1.20 & 0.40 & 0.00 \\ 0.45 & 1.00 & 0.45 \\ 0.00 & 0.40 & 1.20 \end{pmatrix}$$

The eigenvalues of $\mathbf{B}$ is 1.6. Since the eigenvalue is greater than 1, the grammar is inconsistent. Let us see what happens when the probabilities are adjusted in order to make the productions supporting termination more likely

$0.20 : S \rightarrow aSbS$	$0.10 : A \rightarrow aSbBb$	$0.20 : B \rightarrow BabB$
$0.80 : S \rightarrow bAb$	$0.10 : A \rightarrow AbAb$	$0.80 : B \rightarrow abA$
$0.80 : A \rightarrow aba$		

The matrix becomes,

$$\mathbf{B} = \begin{pmatrix} 0.4 & 0.8 & 0.0 \\ 0.1 & 0.2 & 0.1 \\ 0.0 & 0.8 & 0.4 \end{pmatrix}$$

The eigenvalue of $\mathbf{B}$ is now 0.71 and the grammar therefore consistent.

Each element in $\mathbf{B}^k$ contains a mean value of occurrences of nonterminals after k derivations. A *nonterminal expectation matrix* $\mathbf{B}^\infty$ is defined as:

$$\mathbf{B}^\infty = \sum_{k \geq 0} \mathbf{B}^k$$

Since $\mathbf{B}^k$ is summed for all k $\geq$ 0, all derivations have been taken into account in $\mathbf{B}^\infty$. If the grammar is consistent it can be computed as:

$$\mathbf{B}^\infty = (\mathbf{I} - \mathbf{B})^{-1}$$

The mean number of nonterminals rewritten in deriving a terminal string equals the sum of each element in the row of $\mathbf{B}^\infty$ for the starting symbol for the grammar. This value is called the *expected derivation length.*

The product $\mathbf{B}^\infty\mathbf{D}$ gives the *terminal expectation matrix.* An element with index (i,j) denotes the expected number of occurrences of terminal a_j in all derivations starting with nonterminal A_i. The mean number of terminals in a string equals the sum of each element in the row of $\mathbf{B}^\infty\mathbf{D}$ for the starting symbol. This value is also called the *expected string length.*

Example B-4:

Consider a grammar $G_4 = \{N, \Sigma, P, S\}$ with productions:

$$\begin{array}{lll} 0.7 : S \rightarrow aBB & 0.5 : A \rightarrow Bb & 0.6 : B \rightarrow aB \\ 0.1 : S \rightarrow bA & 0.1 : A \rightarrow cA & 0.6 : B \rightarrow bB \\ 0.2 : S \rightarrow c & 0.4 : A \rightarrow c & 0.3 : B \rightarrow cc \end{array}$$

By applying Algorithm B-1 the matrices can be obtained:

$$\mathbf{B} = \begin{pmatrix} 0 & 0.1 & 1.4 \\ 0 & 0.1 & 0.5 \\ 0 & 0 & 0.6 \end{pmatrix}$$

$$\mathbf{D} = \begin{pmatrix} 0.7 & 0.1 & 0.2 \\ 0 & 0.5 & 0.5 \\ 0.6 & 0.1 & 0.6 \end{pmatrix}$$

For $\mathbf{B}$ the rows and columns are ordered as S, A, B. For $\mathbf{D}$ the rows are ordered as S, A, B and the columns as a, b, c.

$\mathbf{B}^\infty$ and $\mathbf{B}^\infty\mathbf{D}$ can be computed.

$$\mathbf{B}^\infty = \begin{pmatrix} 1.0 & 0.1 & 4.8 \\ 0.0 & 1.1 & 1.8 \\ 0.0 & 0.0 & 3.3 \end{pmatrix}$$

$$\mathbf{B}^\infty\mathbf{D} = \begin{pmatrix} 3.6 & 5.0 & 3.1 \\ 1.1 & 0.7 & 1.6 \\ 2.0 & 0.3 & 2.0 \end{pmatrix}$$

The expected derivation length is $1.0 + 0.1 + 4.8 = 5.9$. The expected string length is $3.6 + 5.0 + 3.1 = 11.7$.

The methods available to compute statistical information for the entire derivation process is summarized, on the next page, in Algorithm B-3.

ALGORITHM B.3. Information for the entire derivation process.

Input:

Output:

Method:

A stochastic context free grammar $G = (N, \Sigma, P, S)$,
specified as $N = \{A_1, A_2, \cdots, A_n\}$, $\Sigma = \{a_1, a_2, \cdots, a_m\}$, $S = \{A_1\}$
The production must be of the following form:
$p_i : A_i \rightarrow \beta$, where $\beta \in (N \cup \Sigma)^\star$

Consistency (True/False)
Mean_occ(a_i), Expected string length.
Mean_occ(A_i), Expected derivation length.

```
1)    Apply Algorithm B.1 to obtain the nonterminal and terminal
      first moment matrices $\mathbf{B}$ and $\mathbf{D}$.

2)    / * Check consistency */
      **If** $\mathbf{B}$ is positive regular **then**
          Compute the greatest eigenvalue $\rho$ of $\mathbf{B}$
      **End**
      **If** $\rho \leq 1$ **then**
          Consistent = TRUE
      **Else**
          Consistent = FALSE
      **End**

3)    /* Computing the mean values */
      **If** $\mathbf{B}$ is consistent **then**
          Compute $\mathbf{B}^\infty = (\mathbf{I} - \mathbf{B})^{-1}$
             Mean_occ($A_i$) $= \mathbf{B}^\infty_{1,A_i}$
             Expected derivation length $= \sum_{A_i} \mathbf{B}^\infty_{1,A_i}$
          Compute $\mathbf{B}^\infty \mathbf{D}$
             Mean_occ($a_i$) $= (\mathbf{B}^\infty \mathbf{D})_{1,a_i}$
             Expected string length $= \sum_{a_i} (\mathbf{B}^\infty \mathbf{D})_{1,a_i}$
      **End**
```

Bibliography

[1] A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation, and Compiling*, volume 1: Parsing. Prentice-Hall, 1972.

[2] Edward Altman and Ranan Banerji. Some problems of finite representability. *Information and Control*, 1965.

[3] G. Baron and W. Kuich. The characterization of nonexpansive grammars by rational power series. *Information and Control*, 48:109–118, 1981.

[4] J. Berstel and L. Boasson. Context-free languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B Formal Models and Semantics, pages 61–102. Elsevier Science Publishers, 1990.

[5] B. K. Bhargava and K. S. Fu. Stochastic tree systems for syntactic pattern recognition. In *Proceedings Twelfth Annual Allerton Conference on Circuit and System Theory*, 1974.

[6] Taylor L. Booth and Richard A. Thompson. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22(5):442–450, May 1973.

[7] F. Casacuberta and E. Vidal. A parsing algorithm for weighted grammars and substring recognition. In G. Ferraté et al, editor, *NATO ASI Series, Vol F45, Syntactic and Structural Pattern Recognition.* Springer-Verlag, 1988.

[8] N. Chomsky and M.P. Schützenberger. The algebraic theory of context-free languages. In *Computer Programming and Formal Systems.* North-Holland Publishing Company, 1963.

[9] P. A. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. In *SPIE Vol. 1199 Visual Communications and Image Processing IV*, pages 852 – 863, 1989.

[10] Serge Colombo. *Holomorfic Functions of One Variable.* Gordon and Breach Science Publishers, 1983.

[11] J.H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall LTD, 1971.

[12] Hon-Son Don. Stochastic image segmentation using spatial-temporal context. In *9th International Conference on Pattern Recognition*, pages 342–344. IAPR, 1988.

[13] Hon-Son Don and King-Sun Fu. A parallel algorithm for stochastic image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(5):594–602, September 1986.

[14] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, 1973.

[15] W. Feller. *An Introduction to Probability Theory and Its Applications, Vol 1 (Third Edition).* John Wiley & Sons, 1968.

[16] K. S. Fu. On syntactic pattern recognition and stochastic languages. In Satosi Watanabe, editor, *Frontiers of Pattern Recognition*, pages 113–137. Academic Press, 1972.

[17] K. S. Fu. Stochastic languages for picture analysis. *Computer Graphics and Image Processing*, 2:433–453, 1973.

[18] King-Sun Fu. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, 1982.

[19] K.S. Fu. Stochastic automata, stochastic languages and pattern recognition. *Journal of Cybernetics*, 1(3):31–49, 1971.

[20] K.S. Fu. Stochastic tree languages and their application to picture processing. In P. Krishnaiah, editor, *Proceedings of the 5th International Symposium, Multivariate Analysis*, pages 561–579. North-Holland Publishing Company, 1980.

[21] K.S. Fu. Syntactic image modeling using stochastic tree grammars. *Computer Graphics and Image Processing*, 12:136–152, 1980.

[22] Rafael C. Gonzalez and Michael G. Thomason. *Syntactic Pattern Recognition: An Introduction.* Addison-Wesley Publishing Company, 1978.

[23] I.J. Good. Generalizations to several variables of Lagrange's expansion, with applications to stochastic processes. *Proceedings of the Cambridge Philosophical Society (Mathematical and Physical Sciences*, 56(Part 3):367–380, July 1960.

[24] I.J. Good. The generalization of Lagrange's expansion and the enumeration of trees. *Proceedings of the Cambridge Philosophical Society (Mathematical and Physical Sciences*, 61:499–517, 1965.

[25] Ulf Grenander. Syntax-controlled probabilities. Technical report, Brown University, Division of Applied Mathematics, 1967.

[26] Theodore E. Harris. *The Theory of Branching Processes.* Springer-Verlag, 1963.

[27] Michael A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, 1978.

[28] Micha Hofri. *Probabilistic Analysis of Algorithms.* Springer-Verlag, 1987.

[29] Erwin Kreyszig. *Advanced Engineering Mathematics.* John Wiley & Sons, 1979.

[30] Werner Kuick and Arto Salomaa. *Semirings, Automata, Languages.* Springer-Verlag, 1986.

[31] Harry C. Lee and King-Sun Fu. A stochastic syntax analysis procedure and its application to pattern classification. *IEEE Transactions on Computers*, C-21(7):660–666, July 1972.

[32] H.C. Lee and K.S. Fu. A syntactic pattern recognition system with learning capability. In *Fourth International Symposium on Computer and Information Sciences*, pages 425–449, Dec. 14-16 1972.

[33] M. Lothaire. *Combinatorics on words.* Addison-Wesley Publishing Company, 1983.

[34] Shin-Yee Lu and King-Sun Fu. Stochastic error-correcting syntax analysis for recognition of noisy patterns. *IEEE Transactions on Computers*, C-26(12):1268 – 1276, December 1977.

[35] Rohit J. Parikh. On context-free languages. *Journal of the Association for Computing Machinery*, 13(4):570–581, 1966.

[36] E. Parzen. *Stochastic Processes.* Holden-Day, 1962.

[37] E. Persoon and K.S. Fu. Sequential classification of strings generated by scfg's. *International Journal of Computer and Information Sciences*, 4(3):205–217, 1975.

[38] D. L. Pilling. Commutative regular equations and Parikh's theorem. *Journal of the London Mathematical Society*, 2(6), 1973.

[39] George N. Raney. Functional composition patterns and power series reversion. *Transactions of the American Mathematical Society*, 94:441–451, 1960.

[40] R. Michael Range. *Holomorphic Functions and Integral Representations in Several Complex Variables.* Springer-Verlag, 1986.

[41] A. Salomaa. Formal languages and power series. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B Formal Models and Semantics, pages 103–132. Elsevier Science Publishers, 1990.

[42] Arto Salomaa and Matti Soitola. *Automata-Theoretic Aspects of Formal Power Series.* Springer-Verlag, 1978.

[43] Michael G. Thomason. Stochastic syntax-directed translation schemata for correction of errors in context-free languages. *IEEE Transactions on Computers*, C-24(12):1211 – 1216, December 1975.

[44] Michael G. Thomason. Syntactic pattern recognition: Stochastic languages. In Fu and Young, editors, *Handbook of Pattern Recognition and Image Processing*. Academic Press, 1986.

[45] Michael G. Thomason. Generating functions for stochastic context-free grammars. *International Journal of Pattern Recognition and Artificial Intelligence*, 4(4):553–572, December 1990.

[46] Michael G. Thomason. Lecture notes: Discrete parameter Markov chains in computer science. Technical report, Department of Computer Science, University of Tennessee, Knoxwille, TN 37996 USA, 1992.

[47] Wen-Hsiang Tsai and King-Sun Fu. A pattern deformation model and bayes error-correcting recognition system. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-9(12), December 1979.

[48] Wen-Hsiang Tsai and King Sun Fu. Attributed grammar - a tool for combining syntactic and statistical approaches to pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-10(12):873–885, 1980.

[49] C. S. Wetherell. Probabilistic languages: A review and some open questions. *Computing Surveys*, 12(4):361–379, December 1980.

[50] Herbert S. Wilf. *generatingfunctionology*. Academic Press, Inc., 1990.

[51] Bian Zhaoqi. An approach to string representation of waveforms and its application to waveform recognition. In *Proceedings from 7th. International Conference on Pattern Recognition*, pages 654–656, 1986.