Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

93

Anton Nijholt

Context-Free Grammars: Covers, Normal Forms, and Parsing

Springer-Verlag
Berlin Heidelberg New York

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

93

Anton Nijholt

Context-Free Grammars: Covers, Normal Forms, and Parsing

Springer-Verlag
Berlin Heidelberg New York 1980

Author

Anton Nijholt
Vrije Universiteit
Wiskundig Seminarium
De Boelelaan 1081
Postbus 7161
1007 MC Amsterdam
The Netherlands

AMS Subject Classifications (1979): 68F05, 68F25
CR Subject Classifications (1974): 4.12, 5.23

ISBN 3-540-10245-0 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-10245-0 Springer-Verlag New York Heidelberg Berlin

Library of Congress Cataloging in Publication Data. Nijholt, Anton, 1946-
Context-free grammars. (Lecture notes in computer science; 93) Bibliography: p.
Includes index. 1. Formal languages. I. Title. II. Series.
QA267.3.N54. 511.3. 80-21378

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2145/3140-543210

Lecture Notes in Computer Science

Vol. 1: GI-Gesellschaft für Informatik e.V. 3. Jahrestagung, Hamburg, 8.–10. Oktober 1973. Herausgegeben im Auftrag der Gesellschaft für Informatik von W. Brauer. XI, 508 Seiten. 1973.

Vol. 2: GI-Gesellschaft für Informatik e.V. 1. Fachtagung über Automatentheorie und Formale Sprachen, Bonn, 9.–12. Juli 1973. Herausgegeben im Auftrag der Gesellschaft für Informatik von K.-H. Böhling und K. Indermark. VII, 322 Seiten. 1973.

Vol. 3: 5th Conference on Optimization Techniques, Part I. (Series: I.F.I.P. TC7 Optimization Conferences.) Edited by R. Conti and A. Ruberti. XIII, 565 pages. 1973.

Vol. 4: 5th Conference on Optimization Techniques, Part II. (Series: I.F.I.P. TC7 Optimization Conferences.) Edited by R. Conti and A. Ruberti. XIII, 389 pages. 1973.

Vol. 5: International Symposium on Theoretical Programming. Edited by A. Ershov and V. A. Nepomniaschy. VI, 407 pages. 1974.

Vol. 6: B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, Matrix Eigensystem Routines – EISPACK Guide. XI, 551 pages. 2nd Edition 1974. 1976.

Vol. 7: 3. Fachtagung über Programmiersprachen, Kiel, 5.–7. März 1974. Herausgegeben von B. Schlender und W. Friedinghaus. VI, 225 Seiten. 1974.

Vol. 8: GI-NTG Fachtagung über Struktur und Betrieb von Rechensystemen, Braunschweig, 20.–22. März 1974. Herausgegeben im Auftrag der GI und der NTG von H.-O. Leilich. VI, 340 Seiten. 1974.

Vol. 9: GI-BIFOA Internationale Fachtagung: Informationszentren in Wirtschaft und Verwaltung. Köln, 17./18. Sept. 1973. Herausgegeben im Auftrag der GI und dem BIFOA von P. Schmitz. VI, 259 Seiten. 1974.

Vol. 10: Computing Methods in Applied Sciences and Engineering, Part 1. International Symposium, Versailles, December 17–21, 1973. Edited by R. Glowinski and J. L. Lions. X, 497 pages. 1974.

Vol. 11: Computing Methods in Applied Sciences and Engineering, Part 2. International Symposium, Versailles, December 17–21, 1973. Edited by R. Glowinski and J. L. Lions. X, 434 pages. 1974.

Vol. 12: GFK-GI-GMR Fachtagung Prozessrechner 1974. Karlsruhe, 10.–11. Juni 1974. Herausgegeben von G. Krüger und R. Friehmelt. XI, 620 Seiten. 1974.

Vol. 13: Rechnerstrukturen und Betriebsprogrammierung, Erlangen, 1970. (GI-Gesellschaft für Informatik e.V.) Herausgegeben von W. Händler und P. P. Spies. VII, 333 Seiten. 1974.

Vol. 14: Automata, Languages and Programming – 2nd Colloquium, University of Saarbrücken, July 29–August 2, 1974. Edited by J. Loeckx. VIII, 611 pages. 1974.

Vol. 15: L Systems. Edited by A. Salomaa and G. Rozenberg. VI, 338 pages. 1974.

Vol. 16: Operating Systems, International Symposium, Rocquencourt 1974. Edited by E. Gelenbe and C. Kaiser. VIII, 310 pages. 1974.

Vol. 17: Rechner-Gestützter Unterricht RGU '74, Fachtagung, Hamburg, 12.–14. August 1974, ACU-Arbeitskreis Computer-Unterstützter Unterricht. Herausgegeben im Auftrag der GI von K. Brunnstein, K. Haefner und W. Händler. X, 417 Seiten. 1974.

Vol. 18: K. Jensen and N. E. Wirth, PASCAL – User Manual and Report. VII, 170 pages. Corrected Reprint of the 2nd Edition 1976.

Vol. 19: Programming Symposium. Proceedings 1974. V, 425 pages. 1974.

Vol. 20: J. Engelfriet, Simple Program Schemes and Formal Languages. VII, 254 pages. 1974.

Vol. 21: Compiler Construction, An Advanced Course. Edited by F. L. Bauer and J. Eickel. XIV. 621 pages. 1974.

Vol. 22: Formal Aspects of Cognitive Processes. Proceedings 1972. Edited by T. Storer and D. Winter. V, 214 pages. 1975.

Vol. 23: Programming Methodology. 4th Informatik Symposium, IBM Germany Wildbad, September 25–27, 1974. Edited by C. E. Hackl. VI, 501 pages. 1975.

Vol. 24: Parallel Processing. Proceedings 1974. Edited by T. Feng. VI, 433 pages. 1975.

Vol. 25: Category Theory Applied to Computation and Control. Proceedings 1974. Edited by E. G. Manes. X, 245 pages. 1975.

Vol. 26: GI-4. Jahrestagung, Berlin, 9.–12. Oktober 1974. Herausgegeben im Auftrag der GI von D. Siefkes. IX, 748 Seiten. 1975.

Vol. 27: Optimization Techniques. IFIP Technical Conference. Novosibirsk, July 1–7, 1974. (Series: I.F.I.P. TC7 Optimization Conferences.) Edited by G. I. Marchuk. VIII, 507 pages. 1975.

Vol. 28: Mathematical Foundations of Computer Science. 3rd Symposium at Jadwisin near Warsaw, June 17–22, 1974. Edited by A. Blikle. VII, 484 pages. 1975.

Vol. 29: Interval Mathematics. Procedings 1975. Edited by K. Nickel. VI, 331 pages. 1975.

Vol. 30: Software Engineering. An Advanced Course. Edited by F. L. Bauer. (Formerly published 1973 as Lecture Notes in Economics and Mathematical Systems, Vol. 81) XII, 545 pages. 1975.

Vol. 31: S. H. Fuller, Analysis of Drum and Disk Storage Units. IX, 283 pages. 1975.

Vol. 32: Mathematical Foundations of Computer Science 1975. Proceedings 1975. Edited by J. Bečvář. X, 476 pages. 1975.

Vol. 33: Automata Theory and Formal Languages, Kaiserslautern, May 20–23, 1975. Edited by H. Brakhage on behalf of GI. VIII, 292 Seiten. 1975.

Vol. 34: GI – 5. Jahrestagung, Dortmund 8.–10. Oktober 1975. Herausgegeben im Auftrag der GI von J. Mühlbacher. X, 755 Seiten. 1975.

Vol. 35: W. Everling, Exercises in Computer Systems Analysis. (Formerly published 1972 as Lecture Notes in Economics and Mathematical Systems, Vol. 65) VIII, 184 pages. 1975.

Vol. 36: S. A. Greibach, Theory of Program Structures: Schemes, Semantics, Verification. XV, 364 pages. 1975.

Vol. 37: C. Böhm, λ-Calculus and Computer Science Theory. Proceedings 1975. XII, 370 pages. 1975.

Vol. 38: P. Branquart, J.-P. Cardinael, J. Lewi, J.-P. Delescaille, M. Vanbegin. An Optimized Translation Process and Its Application to ALGOL 68. IX, 334 pages. 1976.

Vol. 39: Data Base Systems. Proceedings 1975. Edited by H. Hasselmeier and W. Spruth. VI, 386 pages. 1976.

Vol. 40: Optimization Techniques. Modeling and Optimization in the Service of Man. Part 1. Proceedings 1975. Edited by J. Cea. XIV, 854 pages. 1976.

Vol. 41: Optimization Techniques. Modeling and Optimization in the Service of Man. Part 2. Proceedings 1975. Edited by J. Cea. XIII, 852 pages. 1976.

Vol. 42: James E. Donahue, Complementary Definitions of Programming Language Semantics. VII, 172 pages. 1976.

Vol. 43: E. Specker und V. Strassen, Komplexität von Entscheidungsproblemen. Ein Seminar. V, 217 Seiten. 1976.

Vol. 44: ECI Conference 1976. Proceedings 1976. Edited by K. Samelson. VIII, 322 pages. 1976.

Vol. 45: Mathematical Foundations of Computer Science 1976. Proceedings 1976. Edited by A. Mazurkiewicz. XI, 601 pages. 1976.

Vol. 46: Language Hierarchies and Interfaces. Edited by F. L. Bauer and K. Samelson. X, 428 pages. 1976.

Vol. 47: Methods of Algorithmic Language Implementation. Edited by A. Ershov and C. H. A. Koster. VIII, 351 pages. 1977.

Vol. 48: Theoretical Computer Science, Darmstadt, March 1977. Edited by H. Tzschach, H. Waldschmidt and H. K.-G. Walter on behalf of GI. VIII, 418 pages. 1977.

PREFACE

This monograph develops a theory of grammatical covers, normal forms and parsing. Covers, formally defined in 1969, describe a relation between the sets of parses of two context-free grammars. If this relation exists then in a formal model of parsing it is possible to have, except for the output, for both grammars the same parser.

Questions concerning the possibility to cover a certain grammar with grammars that conform to some requirements on the productions or the derivations will be raised and answered. Answers to these cover problems will be obtained by introducing algorithms that describe a transformation of an input grammar into an output grammar which satisfies the requirements.

The main emphasis in this monograph is on transformations of context-free grammars to context-free grammars in some normal form. However, not only transformations of this kind will be discussed, but also transformations which yield grammars which have useful parsing properties.

Organization of the monograph

This monograph can be viewed as consisting of four parts.

The first part, Chapters 1 through 3, introduces the cover concept, the motivation of our research, the problems and, moreover, it reviews previous research.

The second part, Chapters 4 through 7, provides cover results for normal form transformations of context-free and regular grammars.

The third part, Chapters 8 through 10, is devoted to cover results for three classes of deterministically parsable grammars, viz. LL(k), strict deterministic and LR(k) grammars. In this part, a discussion of some syntactic aspects of compiler writing systems is included.

The fourth and final part of this monograph consists of Chapters 11 and 12. Chapter 11 contains a detailed discussion on simple chain grammars. Chapter 12 surveys parsing strategies for context-free grammars. In this chapter cover properties of transformations to LL(k) and some other classes of grammars are considered.

A Bibliography and an Index appear at the end of the monograph.

A few sections and notes in this monograph are marked with a star. These starred sections and notes can be skipped without loss of continuity. Some of these starred sections and notes deal with syntax categories and grammar functors. Others deal with technical arguments on parsing at a moment that a reader who is not acquainted with some less conventional ideas of parsing will not grasp their significance.

The sections and notes on syntax categories are included to give the interested reader and the reader who is familiar with these concepts a notion of the differences and the similarities between these concepts and the grammar cover concept.

Moreover, it will become clear that in our grammar cover framework of Chapter 2 we have borrowed from ideas of the grammar functor approach.

We have tried to give full and formal proofs for most of the results which appear in this monograph. Only in those cases that proofs are available in publications elsewhere or in cases that *we* had the idea that a certain result should be clear because of its simplicity or because of what has been proven in the foregoing parts of the monograph, we have omitted a proof or formal detail.

Acknowledgements

Several people have helped me prepare this monograph. I should like to mention particularly Michael A. Harrison of the University of California at Berkeley and Jaco W. de Bakker of the Vrije Universiteit and the Mathematical Centre in Amsterdam for providing time, confidence and for their comments on a first handwritten version of the manuscript. Although not all their suggestions have been incorporated many improvements are due to their comments.

Other people, maybe sometimes unknowingly, did encourage me. Especially I want to mention Derick Wood of McMaster's University at Hamilton.

This monograph was prepared during my stay with the Vakgroep Informatica of the Department of Mathematics of the Vrije Universiteit in Amsterdam. I want to express my gratitude to Marja H., Marja V., Betty and Carla for being there and helping me. Carla Reuvecamp did an excellent job of typing the lengthy manuscript.

Anton Nijholt
April 1980.

CONTENTS

CHAPTER 1

INTRODUCTIONS AND PRELIMINARIES

1.1. INTRODUCTION

Two context-free grammars which generate the same language are said to be weakly equivalent. Weak equivalence can be considered as a relation of grammatical similarity of context-free grammars. If two grammars G_1 and G_2 are weakly equivalent, then for each parse tree T_1 of G_1 there exists a parse tree T_2 of G_2 which has the same frontier, and conversely. Clearly, this relation of weak equivalence does not necessarily say that the shapes of the trees are closely related. Grammatical similarity relations have been introduced which describe relationships between the parse trees of the two grammars.

These relations sometimes but not always presuppose weak equivalence. For example, there exists the relation of structural equivalence. In that case we demand that, except for a relabeling of the internal nodes, the parse trees of the two grammars are the same.

Many other relations have been defined. Only a few will be considered here and only one of them, the grammar cover, will be treated in detail.

In many cases of interest it is quite natural to have weak equivalence between two grammars. For example, a grammar can be changed to an other grammar which generates the same language. Such a transformation on a grammar may be done for several reasons.

By definition, each context-free language is generated by a context-free grammar. Instead of arbitrary context-free grammars one can consider context-free grammars which conform to some requirements on, for example, the productions or the derivations of the grammar. Then it is natural to ask whether each context-free language has a context-free grammar of this form and, if so, how to transform a grammar to this (normal) form.

One reason for considering normal forms may be the inherent mathematical interest in how to generate a family of context-free languages with a grammatical description as simple as possible. Moreover, normal forms can simplify proofs and descriptions in the field of formal languages and parsing. However, in 1975 it still could be remarked (Hotz[65]):

"*Resultate über die strukturelle Verwandschaft verschiedener Sprachen existieren kaum. Selbst bei der Herleitung von Normalformentheoremen für Grammatiken hat man sich mit der Feststellung der schwachen Äquivalenz begnügt*".

Some normal form descriptions for context-free grammars, or for grammars belonging to the various subclasses of the class of context-free grammars, can be particu-

larly amenable for parsing, and this can be a strong motivation to transform grammars.

Transforming grammars into normal forms or to grammars which have other parsing properties can sometimes lead to faster or more compact parsers for these grammars. However, in these cases it is desirable to have a stronger relation than weak equivalence between the original grammar and the newly obtained grammar. This can be seen as follows.

Consider a very practical situation in which we want to build a compiler for a given programming language. We are interested in the part of the compiler which performs the syntactic analysis. We can consider this analysis as a translation from a sentence to a string which consists of procedure calls to perform the code generation.

One now can try to find a 'better' grammar (from the point of view of parsing) such that this translation is preserved. If this is possible, then parsing can be done with respect to the new grammar. The concept of grammar cover which is studied in this monograph describes a preservation of this translation.

We confine ourselves to a model of parsing in which each sentence is given a 'description'of each of its parse trees by means of a string of productions of the grammar. The correspondence of two grammars which is described by the grammar cover is the relation between the parse tree descriptions for a given sentence. In Chapter 8 we have a short discussion on the limitations of this model.

Often a description of a parse tree of a sentence w is given by means of a left or right parse, that is, a string of productions which are used in a derivation (leftmost or rightmost) of the sentence w. Although we will also allow other descriptions of parse trees, it will be clear that we are interested in the relationships among the derivations of sentences of the grammars which we want to relate. This idea can be recognized in many concepts.

In the older literature one can find ideas and examples which come close to later formal concepts. Transformations on context-free grammars have been defined in practically oriented situations of compiler construction. In those cases no general definition of the relation between the grammars was presented.

Grammar covers, in the sense that we will use them here, were introduced about 1969 by Gray and Harrison [48]. Their interest in this concept was based on its applications in the field of parsing.
The product of the syntactic analysis, the parse, can be considered as the argument of a semantic mapping. In the case that a context-free grammar G' covers a context-free grammar G, then each parse with respect to G' of a sentence w can be mapped by a homomorphism on a parse with respect to G of w. Hence, we can parse with respect to G' and use the original semantic mapping.

Other examples of grammatical similarity relations are grammar functors and grammar forms. Grammar functors (X-functors) were introduced by Hotz [63,64] as special functors on categories associated with (general) phrase structure grammars. These

categories originate from work on switching circuits. The objects of a syntax category are strings over the grammar alphabet. The derivations are then considered as morphisms. The main concern has been to find an algebraic framework for describing general properties of phrase structure grammars. Later, functors have been considered from a more practical point of view and topics related to parsing have been discussed within this framework. See, for example, Bertsch [14], Benson [13] and Hotz and Ross [68].

In the case of grammar forms (Cremers and Ginsburg [21]) the starting point is a (master) grammar from which by means of substitutions of the nonterminal and terminal symbols other grammars are obtained. Observations on the parsing properties of the master grammar can be valid for all the grammars in the grammatical family which is obtained by these substitutions (cf. Ginsburg, Leong, Mayer and Wotschke [44]).

There are other examples of grammatical similarity relations. In Hunt and Rosenkrantz [69] many of them are discussed from the point of view of complexity.

In this monograph we will discuss the concept of grammar cover and its usefulness for parsing.

At this point we should mention two approaches which could have been followed and which will not be discussed further.

Firstly, it would be possible to consider transformations on attribute grammars (Knuth [78]). Here, attributes are associated with the nodes of a parse tree. These attributes (which contain the necessary information for the code generation) are obtained from attributes associated with the symbols which appear in the productions and from attribute evaluation rules. If an attribute grammar is transformed to, for example, some normal form attribute grammar, then we have not only the question of language equivalence, but also, explicitly, the question of 'semantic' equivalence. Such an equivalence is explored in Bochmann [15] and Anderson [5].

Secondly, it would have been possible to discuss translation grammars (Brosgol [18]) and transformations on translation grammars.

There is a third remark which we want to make at this point. We consider transformations of grammars. If they are applied with a view to obtain faster or compacter parsing methods then, instead of transforming the grammar, one can build a parser for the grammar and then change (optimize) this parser. This is, for instance, a very common method if an LR-parser is constructed. For example, instead of eliminating single productions from the grammar, single reductions can be eliminated from the parser (cf. e.g. Anderson, Eve and Horning [6]).

Answers to questions on the existence of a covering grammar can be answers to questions whether or not a parser for a given grammar can be modified in certain advantageous ways.

1.2. OVERVIEW OF THE CONTENTS

In Chapters 1 to 6 of this monograph we will be concerned with transformations of arbitrary context-free grammars to context-free grammars in some normal form representation. The main normal forms which will be considered are the non-left-recursive form and the Greibach normal form. Cover results for these normal forms will be presented.

Throughout this monograph we will pay much attention to what has been said before by various authors on these transformations. However, hardly any attention will be paid to grammar functors. Grammar covers are much more amenable than grammar functors and we think this is shown fairly convincingly.

This section will be followed by a section in which we review some basic terminology concerning formal grammars, automata and syntax categories.

In Chapter 2 grammar covers and functors are introduced. The framework for grammar covers which is presented is very general. Partly this is done to obtain an analogy with the grammar functor approach. The second reason, however, is that we need this generality to include various definitions of covers which have been introduced before and to be able to describe practical situations which appear in the field of compiler building.

Chapter 3 shows the efforts which have been made by other authors to grasp some of the 'structure' or 'semantic' preserving properties of transformations of context-free grammars.

In Chapter 4 some general properties of grammar covers are shown and a few preliminary transformations are introduced.

Chapter 5 contains the main transformations of this monograph. It is shown, among others, that any context-free grammar can be covered with a context-free grammar in Greibach normal form. In Chapter 6 we have collected the cover results for normal forms of context-free grammars. Chapter 7 is devoted to some similar results for the class of regular grammars.

In Chapter 8, 9 and 10 we will be concerned with classes of grammars for which there exist parsing methods which can be implemented by a deterministic pushdown transducer. Especially in these chapters we will pay attention to the usefulness of grammar covers for compiler writing systems. Both general cover results and results for normal forms for LL(k), strict deterministic and LR(k) grammars will be presented.

Finally, in Chapter 11 and 12 we discuss a few subclasses of LR(k) grammars in the light of the results which were obtained in the preceeding chapters. In Chapter 11 a variety of results are shown for the class of simple chain grammars. Cover properties, parsing properties and properties of the parse trees of simple chain grammars will be introduced. In Chapter 12 we consider generalizations of the class of simple chain grammars.

1.3. PRELIMINARIES

We review some basic definitions and concepts of formal language theory. Most of the notation used in this monograph is presented in this section. It is assumed that the reader is familiar with the basic results concerning context-free grammars and parsing, otherwise, see Aho and Ullman [3,4], Lewis, Rosenkrantz and Stearns [100] and Harrison [58]. Notations concerning grammars and automata and notations concerning categories follow closely those of Aho and Ullman [3] and Benson [13], respectively.

An *alphabet* is a finite set of *symbols* (equivalently, *letters*). The set of all strings (or *words*) over an alphabet V is denoted by V^*. If $\alpha \in V^*$, then $|\alpha|$, the *length* of α, is the number of occurrences of symbols in α. The *empty* string (the string with length zero) is denoted by ε. If $\alpha \in V^*$, then α^R denotes the *reverse* of α.

The set of non-negative integers is denoted by $\mathbb{N}$. If Q is a set, then $|Q|$ stands for the number of its elements. The *empty set* is denoted by $\emptyset$. If Q and R are sets, then Q\R or Q-R denotes the set $\{x \mid x \in Q \text{ and } x \notin R\}$. V^* is the *free monoid* finitely generated by V. $V^+ = V^* \setminus \{\varepsilon\}$. A (monoid) *homomorphism* is a mapping between monoids with *concatenation* as operation. If V^* and W^* are two free monoids and $h : V^* \to W^*$ is a homomorphism between them, then $h(\varepsilon) = \varepsilon$ and $h(\alpha\beta) = h(\alpha)h(\beta)$ for all $\alpha, \beta \in V^*$.

1.3.1. GRAMMARS, AUTOMATA AND TRANSDUCERS

DEFINITION 1.1. A *context-free grammar* G is a four-tuple $G = (N,\Sigma,P,S)$, where

(i) N and Σ are alphabets, $N \cap \Sigma = \emptyset$ and $S \in N$. The elements of N are called *nonterminals* and those of Σ *terminals*. S is called the *start symbol*.

(ii) P is a finite set of ordered pairs (A,α) such that $A \in N$ and α is a word over the vocabulary $V = N \cup \Sigma$. Elements (A,α) of P are called *productions* and are written $A \to \alpha$.

Context-free grammar will be abbreviated to CFG. Elements of N will generally be denoted by the Roman capitals A, B, C,...; elements of Σ by the smalls a, b, c,... from the first part of the Roman alphabet; X, Y and Z will usually stand for elements of V; elements of Σ^* will be denoted by u, v, w, x, y and z and Greek smalls α, β, γ,... will usually stand for elements of V^*

It will be convenient to provide the productions in P with a *label*. In general these labels will be in a set Δ_G (or Δ if G is understood) and we always take $\Delta_G = \{i \mid 1 \le i \le |P|\}$; we often identify P and Δ_G.

We write $i.A \to \alpha$ if production $A \to \alpha$ has label (or number) i. A is called the *lefthand side* of this production; α is the *righthand side* of the production and α is a *rule alternative* of A. If A has rule alternatives $\alpha_1, \alpha_2, \ldots, \alpha_n$, we write

$$A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$$

hence, '|', a symbol not in V, is used to separate rule alternatives. If these productions have labels $i_1, i_2, \ldots, i_n$, then we use the notation

$$i_1/i_2/\ldots/i_n.\ A \to \alpha_1|\alpha_2|\ldots\ldots|\alpha_n.$$

If $A \in N$, then $rhs(A) = \{\alpha \mid A \to \alpha \text{ is in } P\}$.

DEFINITION 1.2. Let $G = (N,\Sigma,P,S)$ be a CFG. For $\alpha,\beta \in V^*$ we say that α directly *derives* β, written $\alpha \Rightarrow_G \beta$, if there exist $\alpha_1,\alpha_2 \in V^*$ and $A \to \gamma$ in P such that $\alpha = \alpha_1 A \alpha_2$ and $\beta = \alpha_1 \gamma \alpha_2$.
If $\alpha_1 \in \Sigma^*$ we say that α *left derives* β, written $\alpha \underset{L}{\Rightarrow}_G \beta$. If $\alpha_2 \in \Sigma^*$ we say that α *right derives* β, written $\alpha \underset{R}{\Rightarrow}_G \beta$.

The subscript G denoting the grammar in question is omitted whenever the identity of this grammar is clear from context. The *transitive-reflexive* closures of these relations are denoted by $\overset{*}{\Rightarrow}$, $\overset{*}{\underset{L}{\Rightarrow}}$ and $\overset{*}{\underset{R}{\Rightarrow}}$, respectively. The *transitive-irreflexive* closures are denoted by $\overset{+}{\Rightarrow}$, $\overset{+}{\underset{L}{\Rightarrow}}$ and $\overset{+}{\underset{R}{\Rightarrow}}$, respectively.

A sequence $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots\ldots \Rightarrow \alpha_n$ is called a *derivation* of α_n from α_0. A sequence $\alpha_0 \underset{L}{\Rightarrow} \alpha_1 \underset{L}{\Rightarrow} \ldots\ldots \underset{L}{\Rightarrow} \alpha_n$ ($\alpha_0 \underset{R}{\Rightarrow} \alpha_1 \underset{R}{\Rightarrow} \ldots\ldots \underset{R}{\Rightarrow} \alpha_n$) is called a *leftmost (rightmost)* derivation of α_n from α_0.

If we want to indicate a derivation using a specific sequence π of productions, we write $\overset{\pi}{\Rightarrow}$ ($\overset{\pi}{\underset{L}{\Rightarrow}}$, $\overset{\pi}{\underset{R}{\Rightarrow}}$), hence, $\pi \in P^*$ or $\pi \in \Delta^*$. In some cases we will use the notation $\alpha \overset{n}{\Rightarrow} \beta$ ($\alpha \overset{n}{\underset{L}{\Rightarrow}} \beta$, $\alpha \overset{n}{\underset{R}{\Rightarrow}} \beta$) to indicate that the derivation in question is such that α derives β in n steps, that is, $(\alpha,\beta) \in (\Rightarrow)^n$.

DEFINITION 1.3. Let $G = (N,\Sigma,P,S)$ be a CFG. The *language* of G is the set $L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$. For any $\alpha \in V^*$, $L(\alpha) = \{w \in \Sigma^* \mid \alpha \overset{*}{\Rightarrow} w\}$. CFG G is said to be *unambiguous* if there does not exist $w \in \Sigma^*$ and $\pi, \pi' \in \Delta^*$ such that $S \overset{\pi}{\underset{L}{\Rightarrow}} w$ and $S \overset{\pi'}{\underset{L}{\Rightarrow}} w$, where $\pi \neq \pi'$. Otherwise, G is said to be *ambiguous*. Let $w \in L(G)$, then w is called a sentence of G. L(G) is said to be a *context-free language* (CFL for short).

DEFINITION 1.4. Let $G = (N,\Sigma,P,S)$ be a CFG. Let $\alpha \in V^*$.

a. $k : \alpha$ is the *prefix* of α with length k if $|\alpha| \geq k$, otherwise $k : \alpha = \alpha$.
b. $\alpha : k$ is the *suffix* of α with length k if $|\alpha| \geq k$, otherwise $\alpha : k = \alpha$.
c. $FIRST_k(\alpha) = \{k : w \in \Sigma^* \mid \alpha \overset{*}{\Rightarrow} w\}$.

Index k of $FIRST_k$ will be omitted when $k = 1$.

NOTATION 1.1. Let Σ and Δ be disjoint alphabets. Homomorphism $h_\Sigma : (\Sigma \cup \Delta)^* \to \Delta^*$ is defined by

$h_\Sigma(X) = X$ if $X \in \Delta$, and

$h_\Sigma(X) = \varepsilon$ if $X \in \Sigma$.

Homomorphism h_Σ will be called the *Σ-erasing homomorphism.*

The number of different leftmost derivations from S to w is called the *degree of ambiguity* of w (with respect to G), written <w,G>. By convention, if $w \notin L(G)$, then <w,G> = 0. We say that $\alpha \in V^*$ is a *sentential form,* a *left sentential form* or a *right sentential form,* if $S \overset{*}{\Rightarrow} \alpha$, $S \overset{*}{\underset{L}{\Rightarrow}} \alpha$ and $S \overset{*}{\underset{R}{\Rightarrow}} \alpha$, respectively.

Derivations (or rather, equivalence classes of derivations) can be represented by *trees*. We distinguish between *derivation trees* and *parse trees*.

DEFINITION 1.5. A *derivation tree* is recursively defined by

(i) A single node labeled S is a derivation tree.

(ii) For every derivation tree, let D, labeled $A \in N$, be a leaf of the tree. If $A \to X_1X_2\ldots X_n$ ($X_i \in V$, $1 \le i \le n$) is in P, the tree obtained by appending to D n sons with labels $X_1, X_2, \ldots, X_n$ in order from the left, is a derivation tree. If $A \to \varepsilon$ is in P, the tree obtained by appending to D one son with label ε is a derivation tree.

The set PTR(G), the set of *parse trees* of G, consists of all derivation trees where each leaf is labeled with a terminal or with ε. The *frontier* of a derivation tree is the string obtained by concatenating the labels of the leaves from left to right. If T is a derivation tree, then fr(T) denotes the frontier of T.

DEFINITION 1.6.

a. Let G = (N,Σ,P,S) be a CFG. Define $P' = \{A \to [\alpha] \mid A \to \alpha \in P\}$, where '[' and ']' are special brackets that are not terminal symbols of G. [G] = (N,Σ ∪ {[,]},P',S), the *parenthesized version* of G, is called a *parenthesis grammar* (McNaughton[107]).

b. Let G = (N,Σ,P,S) be a CFG. Define $P' = \{A \to [_i\alpha]_i \mid i.A \to \alpha \in P\}$, where '$[_i$' and '$]_i$' are special brackets that are not terminal symbols of G. Grammar G_B = $= (N,\Sigma \cup \{[_i \mid i \in \Delta_G\} \cup \{]_i \mid i \in \Delta_G\}, P', S)$, the *bracketed version* of G, is called a *bracketed grammar* (Ginsburg and Harrison [43]).

DEFINITION 1.7.

a. CFG G and CFG H are said to be *weakly equivalent* if L(G) = L(H).

b. CFG G and CFG H are said to be *strongly equivalent* if PTR(G) = PTR(H).

c. CFG G and CFG H are said to be *structurally equivalent* if L([G]) = L([H]).

A symbol $X \in V$ is *useless* in a CFG G = (N,Σ,P,S) with $P \neq \emptyset$, if there does not exist a derivation $S \overset{*}{\Rightarrow} wXy \overset{*}{\Rightarrow} wxy$, where $wxy \in \Sigma^*$. There exists a simple algorithm to remove all useless symbols from a CFG (Aho and Ullman [3]). Throughout this monograph we assume that the grammars under consideration have no useless symbols. Any production of the form $A \to \alpha$ with $\alpha \in N$ is called a *single production.*

DEFINITION 1.8. A CFG $G = (N,\Sigma,P,S)$ is

a. *reduced*, if it has no useless symbols or if $P = \emptyset$.
b. *ε-free*, if $P \subseteq N \times V^+$ or $P \subseteq N \times (V\backslash\{S\})^+ \cup \{S \to \varepsilon\}$.
c. *cycle-free*, if, for any $A \in N$, a derivation $A \overset{+}{\Rightarrow} A$ is not possible.
d. *proper*, if G has no useless symbols, G is ε-free and G is cycle-free.

DEFINITION 1.9. Let $G = (N,\Sigma,P,S)$ be a CFG. A nonterminal $A \in N$ is said to be *left recursive* if there exists $\alpha \in V^*$ such that $A \overset{+}{\Rightarrow} A\alpha$. Grammar G is said to be *left recursive* if there exists a left recursive nonterminal in N. Otherwise, G is said to be *non-left-recursive* (NLR).

For any CFG $G = (N,\Sigma,P,S)$ define $G^R = (N,\Sigma,P^R,S)$ with $P^R = \{A \to \alpha^R \mid A \to \alpha \in P\}$. A CFG G is said to be *non-right-recursive* (NRR) if G^R is NLR.

DEFINITION 1.10. A CFG $G = (N,\Sigma,P,S)$ is

a. in *Greibach normal form* (GNF) if

$$P \subseteq N \times \Sigma N^* \text{ or } P \subseteq N \times \Sigma(N\backslash\{S\})^* \cup \{S \to \varepsilon\}.$$

b. in *quasi Greibach normal form* (quasi-GNF) if

$$P \subseteq N \times \Sigma V^* \text{ or } P \subseteq N \times \Sigma(V\backslash\{S\})^* \cup \{S \to \varepsilon\}.$$

c. *left factored* if P does not contain distinct productions of the form $A \to \alpha\beta_1$ and $A \to \alpha\beta_2$ with $\alpha \neq \varepsilon$.

We say that G is in $\overline{\text{GNF}}$ if grammar G^R is in GNF . For each CFL one can find a CFG which is in one of the forms defined in the Definitions 1.8 to 1.10. Greibach normal form is also called *standard form*. A grammar is said to be in *standard 2-form* if it is in GNF and each righthand side of a production contains at most two nonterminals.

DEFINITION 1.11. A CFG $G = (N,\Sigma,P,S)$ is said to be

a. *right regular*, if each production in P is of the form $A \to aB$ or $A \to a$, where $A,B \in N$ and $a \in \Sigma$.
b. *left regular*, if each production in P is of the form $A \to Ba$ or $A \to a$, where $A,B \in N$ and $a \in \Sigma$.

A *regular* grammar is a grammar which is either left regular or right regular. A set L is said to be regular if there exists a regular grammar G such that $L = L(G)$.

Now we will generalize grammars to (simple) syntax directed translation schemes.