EATCS
Monographs on Theoretical Computer Science
Volume 20

Editors: W. Brauer G. Rozenberg A. Salomaa

Seppo Sippu
Eljas Soisalon-Soininen

Parsing Theory

Volume II
LR(k) and LL(k) Parsing

Springer-Verlag
世界图书出版公司

Seppo Sippu
Eljas Soisalon-Soininen

Parsing Theory

Volume II
LR(k) and LL(k) Parsing

With 110 Figure

江苏工业学院图书馆
藏书章

Springer-Verlag Berlin Heidelberg NewYork
London Paris Tokyo Hong Kong Barcelona
世界图书出版公司 北京·广州·上海·西安

Authors
Professor S. Sippu
Department of Computer Science, University of Jyväskylä
Seminaarinkatu 15, SF-40100 Jyväskylä, Finland

Professor E. Soisalon-Soininen
Department of Computer Science, University of Helsinki
Teollisuuskatu 23, SF-00510 Helsinki, Finland

Editors
Prof. Dr. Wilfried Brauer
Institut für Informatik, Technische Universität München
Arcisstr. 21, D-8000 München 2, Germany

Prof. Dr. Grzegorz Rozenberg
Institute of Applied Mathematics and Computer Science
University of Leiden, Niels-Bohr-Weg 1, P.O. Box 9512
NL-2300 RA Leiden, The Netherlands

Prof. Dr. Arto Salomaa
Department of Mathematics, University of Turku
SF-20500 Turku 50, Finland

ISBN 3-540-51732-4 Springer-Verlag Berlin Heidelberg NewYork
ISBN 0-387-51732-4 Springer-Verlag NewYork Berlin Heidelberg

Library of Congress Cataloging-in-Publication Data
(Revised for vol. 2)
Sippu, Seppo, 1950-
Parsing theory.
(EATCS monographs on theoretical computer science ; v. 15, 20)
Includes bibliographical indexes.
Contents: v. 1. Languages and parsing -- v. 2. LR(*k*) and LL(*k*) parsing.
1. Parsing (Computer grammar) 2. Formal languages.
I. Soisalon-Soininen, Eljas, 1949- . II. Title.
III. Series: EATCS monographs on theoretical computer science ; v. 15, etc.
QA267.3.S59 1988 511.3 88-20091
ISBN 0-387-13720-3 (U.S. : v. 1)
ISBN 0-387-51732-4 (U.S. : v. 2)

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in other ways, and storage in data banks. Duplication of this publication or parts thereof is only permitted under the provisions of the German Copyright Law of September 9, 1965, in its current version, and a copyright fee must always be paid. Violations fall under the prosecution act of the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1990

Reprinted by World Publishing Corporation, Beijing, 1993
for distribution and sale in The People's Republic of China only
ISBN 7 - 5062 - 1530 - 6

Preface

This work is Volume II of a two-volume monograph on the theory of deterministic parsing of context-free grammars. Volume I, "Languages and Parsing" (Chapters 1 to 5), was an introduction to the basic concepts of formal language theory and context-free parsing. Volume II (Chapters 6 to 10) contains a thorough treatment of the theory of the two most important deterministic parsing methods: LR(k) and LL(k) parsing. Volume II is a continuation of Volume I; together these two volumes form an integrated work, with chapters, theorems, lemmas, etc. numbered consecutively.

Volume II begins with Chapter 6 in which the classical constructions pertaining to LR(k) parsing are presented. These include the canonical LR(k) parser, and its reduced variants such as the LALR(k) parser and the SLR(k) parser. The grammar classes for which these parsers are deterministic are called LR(k) grammars, LALR(k) grammars and SLR(k) grammars; properties of these grammars are also investigated in Chapter 6. A great deal of attention is paid to the rigorous development of the theory: detailed mathematical proofs are provided for most of the results presented.

Chapter 7 is devoted to the construction and implementation of LR(k) parsers using lookahead length $k = 1$. Efficient algorithms are presented for computing parsing tables for SLR(1), canonical LR(1) and LALR(1) parsers. Special attention is paid to the optimization of LR(1) parsers. An efficient general algorithm is presented for eliminating reductions by unit rules from an LR(1) parsing table. In developing these algorithms, substantial use is made of the results of Volume I, Chapter 2, where a general algorithm for evaluating a binary relational expression was presented.

Chapter 8 deals with the theory of LL(k) parsing. The constructions pertaining to LL(k) parsing are developed in a way analogous to that used in LR(k) parsing, so as to expose the dualism between these theories. For example, canonical LL(k) parsers and LALL(k) parsers are defined as counterparts of the canonical LR(k) and LALR(k) parsers. The relationship between the LL(k) and LR(k) grammars is studied in detail, and methods for transforming subclasses of LR(k) grammars into LL(k) grammars are presented.

Chapter 9 deals with the problem of syntax error handling in parsers. The nature of syntax errors is discussed, and algorithms for constructing error recovery routines for LL(1) and LR(1) parsers are presented. The treatment in this chapter is somewhat less formal than in the other chapters.

Volume II concludes with Chapter 10 that deals with the complexity of testing whether or not a given context-free grammar belongs to one of the grammar classes discussed in the previous chapters. Efficient polynomial-time algorithms are developed for LR(k), SLR(k), LL(k) and SLL(k) testing when the lookahead length k is fixed. Hardness results are derived for the case in which k is a parameter of the testing problems. Upper and lower bounds on the complexity of LALR(k) and LALL(k) testing are also established.

Jyväskylä and Helsinki, June 1990

Seppo Sippu
Eljas Soisalon-Soininen

Acknowledgements
The work was supported by the Academy of Finland, the Finnish Cultural Foundation, and the Ministry of Education of Finland.

此为试读,需要完整PDF请访问: www.ertongbook.com

Contents

Contents of Volume I

6. LR(k) Parsing

In this chapter we shall consider a general method for deriving deterministic right parsers for context-free grammars. The method will be called "LR(k) parsing". The acronym "LR(k)" refers to the most general deterministic parsing method in which the input string is parsed (1) in a single Left-to-right scan, (2) producing a Right parse, and (3) using lookahead of length k.

LR(k) parsers are a generalization of the nondeterministic shift-reduce parser presented in Section 5.2 and of the simple precedence parser presented in Section 5.7. The key idea in the generalization is that the stack symbols, which in the shift-reduce and simple precedence parsers are plain grammar symbols, are divided up into one or more "context-dependent" symbols. For each grammar symbol X there will be as many stack symbols as there are distinct equivalence classes of the form $[\gamma X]$, where γX is a stack string of the shift-reduce parser. Here two stack strings $\gamma_1 X$ and $\gamma_2 X$ are called equivalent if (to put it informally) exactly the same set of parsing actions are valid in the contexts $\gamma_1 X$ and $\gamma_2 X$. In this way, replacing symbols X by equivalence classes $[\gamma X]$, we can restrict the applicability of the actions of the shift-reduce parser so that a deterministic right parser is obtained for a large subclass of the context-free grammars. These grammars, called the "LR(k) grammars", form a powerful means of language description: any deterministic language (i.e., a language accepted by a deterministic pushdown automaton) can be generated by an LR(k) grammar.

In Section 6.1 we shall study the properties of the stack strings of the shift-reduce parser. The stack strings that appear in the stack in accepting computations will be called "viable prefixes". In any grammar, the viable prefixes form a regular language over the alphabet of the grammar. In Section 6.2 we shall present, for natural number k, an equivalence relation on the set of viable prefixes. This relation, called "LR(k)-equivalence", is obtained via sets of "valid k-items". The k-items are a generalization of the grammar positions used in Section 5.5 in constructing strong LL(1) parsers. The LR(k)-equivalence is of finite index, that is, it has only a finite number of distinct equivalence classes. Any equivalence class can be represented by a certain set of valid k-items. Moreover, it is possible to compute these sets from the grammar.

In Section 6.3 we shall use the concept of LR(k)-equivalence to define the general notion of an LR(k) parser, called the "canonical LR(k) parser". This is a right parser which uses k-length lookahead strings and whose stack strings consist of equivalence classes of viable prefixes. In Section 6.4 we shall study the properties of LR(k) grammars. In Sections 6.5 and 6.6 we shall consider some practical

variations of the canonical **LR(k) parser**. These variations are called "LALR(k) parsers", "LA(k)LR(l) parsers", and "SLR(k) parsers". The classes of grammars for which these parsers are deterministic are called, respectively, "LALR(k) grammars", "LA(k)LR(l) grammars", and "SLR(k) grammars". For all $k \geqslant 0$, these classes are contained in the class of LR(k) grammars. The smallest of these classes, the class of SLR(k) grammars, is powerful enough to generate all deterministic languages. The chapter concludes with Section 6.7, in which we shall show that any LR(k) grammar can be transformed into an equivalent LR(1) grammar. This means that any deterministic language can in fact be generated by an SLR(1) grammar.

6.1 Viable Prefixes

We begin by considering the problem of constructing a deterministic right parser for the grammar G_{ab}:

$$S \to aA|bB \ ,$$
$$A \to c|dAd \ ,$$
$$B \to c|dBd \ .$$

G_{ab} is an s-grammar and generates the language

$$L(G_{ab}) = \{a, b\}\{d^n c d^n | n \geqslant 0\} \ .$$

As G_{ab} has two rules with the same right-hand side, it is not a simple precedence grammar, and so its simple precedence parser is nondeterministic. In fact, to any configuration of the form

$$\$\alpha c \,\mathbf{I}\, y\$, \quad \text{where } \$\alpha{:}1 \in \{a, b, d\} \quad \text{and} \quad 1{:}y\$ \in \{\$, d\} \ ,$$

applies a reduce action by both $A \to c$ and $B \to c$.

We might try to make the parser deterministic by extending the lookahead and lookback symbols of the reduce actions into strings of length k, for some sufficiently great k. This would result in a parser in which the reduce actions by $A \to c$ and $B \to c$ are of the forms

$$\alpha c \,\mathbf{I}\, x \to \alpha A \,\mathbf{I}\, x, \qquad \beta c \,\mathbf{I}\, y \to \beta B \,\mathbf{I}\, y \ ,$$

where α and β are certain strings in $\$V^*{:}k$ and x and y are certain strings in $k{:}T^*\$$. But then, in particular, there would be the pair of actions

$$d^k c \,\mathbf{I}\, d^k \to d^k A \,\mathbf{I}\, d^k, \qquad d^k c \,\mathbf{I}\, d^k \to d^k B \,\mathbf{I}\, d^k \ .$$

This is because some reduce action by $A \to c$ must be applicable to the configuration $\$ad^k c \,\mathbf{I}\, d^k\$$ if the sentence $ad^k c d^k$ is to be accepted, and some reduce

action by $B \to c$ must be applicable to the configuration $\$bd^kc \mid d^k\$$ if the sentence bd^kcd^k is to be accepted. As the pair of actions exhibits a reduce-reduce conflict, we must conclude that it is impossible to obtain a deterministic right parser for G_{ab} just by adding lookahead and lookback strings to the actions of the shift-reduce parser.

To solve the problem we take a closer look at those strings that can appear in the stack in some accepting computation of the shift-reduce parser. We call these strings *viable stack strings*. In general, a string γ is a viable stack string of a pushdown automaton M if

$$\$\gamma_s \mid w\$ \Rightarrow^* \$\gamma \mid y\$ \Rightarrow^* \$\gamma_f \mid \$ \qquad \text{in } M$$

for some input strings w and y and final stack contents γ_f. (γ_s is the initial stack contents of M.)

Obviously, the set of viable stack strings of the shift-reduce parser for G_{ab} is

$$\begin{aligned}&\{\varepsilon\} \cup \{ad^n \mid n \geqslant 0\} \cup \{ad^nc \mid n \geqslant 0\}\\ &\quad \cup \{ad^nA \mid n \geqslant 0\} \cup \{ad^nAd \mid n \geqslant 1\}\\ &\quad \cup \{bd^n \mid n \geqslant 0\} \cup \{bd^nc \mid n \geqslant 0\}\\ &\quad \cup \{bd^nB \mid n \geqslant 0\} \cup \{bd^nBd \mid n \geqslant 1\}\\ &\quad \cup \{S\} \ .\end{aligned}$$

To each stack string a number of parsing actions are applicable. However, only few of these yield a viable stack string as a result. For example, to the stack strings ad^nc and bd^nc a reduce action by both $A \to c$ and $B \to c$ is applicable, but among the resulting strings ad^nA, bd^nB, ad^nB, bd^nA only the first two are viable stack strings. This means that we can resolve the reduce-reduce conflict between $A \to c$ and $B \to c$ by imposing the additional restriction that a parsing action can be applied only if it is "valid" in that it yields a viable stack string as a result.

In general, we say that an action r of a pushdown automaton M is *valid* for viable stack string γ of M if

$$\$\gamma \mid y\$ \overset{r}{\Rightarrow} \$\gamma' \mid y'\$ \qquad \text{in } M$$

for some input strings y and y' and viable stack string γ'.

As the set of viable stack strings is usually infinite, as is the case in G_{ab}, the reader might feel that it is impossible, in general, to find out which actions are valid for which stack strings. However, we can always divide the set of viable stack strings into a finite number of equivalence classes. Two stack strings belong to the same equivalence class if they have the same set of valid actions. Since for any grammar $G = (V, T, P, S)$ the shift-reduce parser has $|T| + |P| \leqslant |G|$ distinct actions, the number of distinct equivalence classes is bounded by $2^{|G|}$, the number of distinct subsets of a $|G|$-element set.

In the case of G_{ab} the equivalence classes and the associated valid actions are:

equivalence class:	valid actions:
$\{\varepsilon\}$	shift a, shift b
$\{ad^n \mid n \geqslant 0\} \cup \{bd^n \mid n \geqslant 0\}$	shift c, shift d
$\{ad^n c \mid n \geqslant 0\}$	reduce by $A \to c$
$\{aA\}$	reduce by $S \to aA$
$\{ad^n A \mid n \geqslant 1\} \cup \{bd^n B \mid n \geqslant 1\}$	shift d
$\{ad^n Ad \mid n \geqslant 1\}$	reduce by $A \to dAd$
$\{bd^n c \mid n \geqslant 0\}$	reduce by $B \to c$
$\{bB\}$	reduce by $S \to bB$
$\{bd^n Bd \mid n \geqslant 1\}$	reduce by $B \to dBd$
$\{S\}$	—

The idea is to use these equivalence classes as stack symbols of the parser. In the actions of the parser, any grammar symbol X originally located to the left of the delimiter $\mid$ is replaced by an equivalence class of the form $[\delta X]$, where δX is a viable stack string. Accordingly, for each viable stack string δ and terminal a there is the shift action

$$(sa) \qquad [\delta] \mid a \to [\delta][\delta a] \mid ,$$

provided that δa is a viable stack string. Similarly, for each stack string δ and rule $A \to X_1 \ldots X_n$, where each X_i is a single symbol, $1 \leqslant i \leqslant n$, there is the reduce action

$$(ra) \qquad [\delta][\delta X_1] \ldots [\delta X_1 \ldots X_n] \mid \to [\delta][\delta A] \mid ,$$

provided that $\delta X_1, \ldots, \delta X_1 \ldots X_n$, and δA are all viable stack strings. (In the general case, the action may also contain a lookahead string; this will be considered later.) The initial stack contents of the parser are $[\varepsilon]$, and the final stack contents are $[\varepsilon][S]$.

For example, the parser obtained in this way for G_{ab} has, among others, the shift actions

$$[\varepsilon] \mid a \to [\varepsilon][ad^* \cup bd^*] \mid \qquad \text{(shift } a\text{)} ,$$

$$[\varepsilon] \mid b \to [\varepsilon][ad^* \cup bd^*] \mid \qquad \text{(shift } b\text{)} ,$$

and the reduce actions

$$[ad^* \cup bd^*][ad^*c] \mid \to [ad^* \cup bd^*][ad^+A \cup bd^+B] \mid$$
$$\text{(reduce by } A \to c\text{)} ,$$

此为试读,需要完整PDF请访问: www.ertongbook.com

$$[ad^* \cup bd^*][bd^*c]\mid \rightarrow [ad^* \cup bd^*][ad^+A \cup bd^+B]\mid$$
(reduce by $B \rightarrow c$) .

Here we have used regular expressions, rather than single members, to denote the equivalence classes. For regular expression E, $[E]$ means the equivalence class of any w in $L(E)$, that is, $[E] = [w]$ for all w in $L(E)$. Thus we have always $L(E) \subseteq [E]$. In fact we usually have $L(E) = [E]$, as is the case above. Soon we shall see that the equivalence classes are indeed regular languages, for any grammar, and can therefore always be denoted by regular expressions.

Obviously, there is no conflict between the above two reduce actions. Unfortunately, our construction has introduced some new conflicts not present in the original parser. There is a reduce-reduce conflict between two reduce actions by $A \rightarrow c$ and a reduce-reduce conflict between two reduce actions by $B \rightarrow c$. These actions (which conflict with the above two reduce actions) are:

$$[ad^* \cup bd^*][ad^*c]\mid \rightarrow [ad^* \cup bd^*][aA]\mid$$
(reduce by $A \rightarrow c$) ,

$$[ad^* \cup bd^*][bd^*c]\mid \rightarrow [ad^* \cup bd^*][bB]\mid$$
(reduce by $B \rightarrow c$) .

Note that $[aA] \neq [ad^+A \cup bd^+B] \neq [bB]$. Moreover, there is an entirely new type of conflict, a "shift-shift conflict", between the actions

$$[ad^* \cup bd^*]\mid c \rightarrow [ad^* \cup bd^*][ad^*c]\mid \quad \text{(shift } c\text{)} ,$$
$$[ad^* \cup bd^*]\mid c \rightarrow [ad^* \cup bd^*][bd^*c]\mid \quad \text{(shift } c\text{)} ,$$

as well as between the actions

$$[ad^+A \cup bd^+B]\mid d \rightarrow [ad^+A \cup bd^+B][ad^+Ad]\mid \quad \text{(shift } d\text{)} ,$$
$$[ad^+A \cup bd^+B]\mid d \rightarrow [ad^+A \cup bd^+B][bd^+Bd]\mid \quad \text{(shift } d\text{)} .$$

The reason for these new conflicts is that the division into equivalence classes is not refined enough. Consider, for example, the viable stack strings ad^n and ad^nA, $n \geq 0$. For all $n \geq 0$, the strings ad^n belong to the same equivalence class, $[ad^* \cup bd^*]$. However, the strings ad^nA, $n \geq 0$, are divided into two distinct equivalence classes: $[aA]$ and $[ad^+A \cup bd^+B]$. Similarly, bB is not equivalent to bd^nB, $n \geq 1$, although all bd^n, $n \geq 0$, are equivalent. This is an anomaly, because if two stack strings δ_1 and δ_2 are already equivalent it is natural to assume that they remain equivalent if they are lengthened by the same symbol X, to viable stack strings $\delta_1 X$ and $\delta_2 X$. In other words, the equivalence should be *right-invariant*.

Another natural requirement, closely related to right-invariance, is that two equivalent stack strings γ_1 and γ_2 should end with the same symbol, that is, $\gamma_1 : 1 = \gamma_2 : 1$. Observe that otherwise it is not clear how we can define the value of the output effect τ in the case of a reduce action

$$[\delta][\delta X_1] \ldots [\delta X_1 \ldots X_n]\mid \rightarrow [\delta][\delta A]\mid .$$

We can map this action to the rule $A \to X_1 \ldots X_n$ only if the rule is uniquely defined, that is, if there is no other rule $A' \to X_1' \ldots X_n'$ satisfying $[\delta A'] = [\delta A]$, $[\delta X_1'] = [\delta X_1]$, $\ldots$, $[\delta X_1' \ldots X_n'] = [\delta X_1 \ldots X_n]$. Uniqueness is clearly guaranteed if equivalent stack strings γ_1 and γ_2 always satisfy the condition $\gamma_1 : 1 = \gamma_2 : 1$.

To fulfil the above two requirements in the case of the grammar G_{ab}, we must refine the original equivalence as follows:

(1) The class $[ad^* \cup bd^*]$ is split into the classes $[a]$, $[ad^+]$, $[b]$, $[bd^+]$.
(2) The class $[ad^+A \cup bd^+B]$ is split into the classes $[ad^+A]$ and $[bd^+B]$.

The classes under the refined equivalence are represented in Figure 6.1 as nodes of a transition graph. The graph has an edge labeled by symbol X from node $[\delta]$ to node $[\delta X]$ whenever δ and δX are viable stack strings. The graph can be interpreted as a finite automaton, with $[\varepsilon]$ as the initial state and a given class $[\gamma]$ as the only final state. The language accepted by that automaton equals $[\gamma]$.

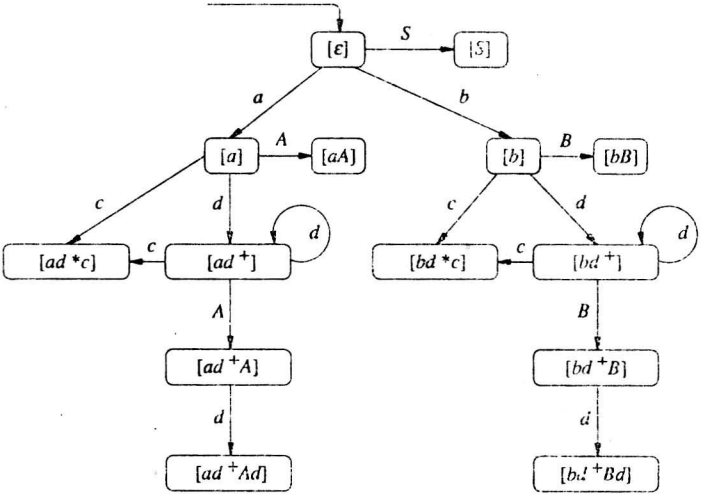

Figure 6.1 Transition graph for the viable stack strings of the shift-reduce parser for the grammar G_{ab}: $S \to aA|bB$, $A \to c|dAd$, $B \to c|dBd$

We are now ready to write down the actions of the parser for G_{ab}. The shift actions are:

$$r_1 = [\varepsilon]\,\mathfrak{f}\,a \to [\varepsilon][a]\,\mathfrak{f}, \qquad \tau(r_1) = \varepsilon\,.$$

$$r_2 = [\varepsilon]\,\mathfrak{f}\,b \to [\varepsilon][b]\,\mathfrak{f}, \qquad \tau(r_2) = \varepsilon\,.$$

$$r_3 = [a]\,\mathfrak{f}\,c \to [a][ad^*c]\,\mathfrak{f}. \qquad \tau(r_3) = \varepsilon\,.$$

$$r_4 = [a]\,\|\,d \to [a][ad^+]\,\|, \qquad \tau(r_4) = \varepsilon .$$

$$r_5 = [ad^+]\,\|\,c \to [ad^+][ad^*c]\,\|, \qquad \tau(r_5) = \varepsilon .$$

$$r_6 = [ad^+]\,\|\,d \to [ad^+][ad^+]\,\|, \qquad \tau(r_6) = \varepsilon .$$

$$r_7 = [ad^+A]\,\|\,d \to [ad^+A][ad^+Ad]\,\|, \qquad \tau(r_7) = \varepsilon .$$

$$r_8 = [b]\,\|\,c \to [b][bd^*c]\,\|, \qquad \tau(r_8) = \varepsilon .$$

$$r_9 = [b]\,\|\,d \to [b][bd^+]\,\|, \qquad \tau(r_9) = \varepsilon .$$

$$r_{10} = [bd^+]\,\|\,c \to [bd^+][bd^*c]\,\|, \qquad \tau(r_{10}) = \varepsilon .$$

$$r_{11} = [bd^+]\,\|\,d \to [bd^+][bd^+]\,\|, \qquad \tau(r_{11}) = \varepsilon .$$

$$r_{12} = [bd^+B]\,\|\,d \to [bd^+B][bd^+Bd]\,\|, \qquad \tau(r_{12}) = \varepsilon .$$

The reduce actions are:

$$r_{13} = [a][ad^*c]\,\| \to [a][aA]\,\|, \qquad \tau(r_{13}) = A \to c .$$

$$r_{14} = [ad^+][ad^*c]\,\| \to [ad^+][ad^+A]\,\|, \qquad \tau(r_{14}) = A \to c .$$

$$r_{15} = [a][ad^+][ad^+A][ad^+Ad]\,\| \to [a][aA]\,\|, \qquad \tau(r_{15}) = A \to dAd .$$

$$r_{16} = [ad^+][ad^+][ad^+A][ad^+Ad]\,\| \to [ad^+][ad^+A]\,\|, \qquad \tau(r_{16}) = A \to dAd .$$

$$r_{17} = [\varepsilon][a][aA]\,\| \to [\varepsilon][S]\,\|, \qquad \tau(r_{17}) = S \to aA .$$

$$r_{18} = [b][bd^*c]\,\| \to [b][bB]\,\|, \qquad \tau(r_{18}) = B \to c .$$

$$r_{19} = [bd^+][bd^*c]\,\| \to [bd^+][bd^+B]\,\|, \qquad \tau(r_{19}) = B \to c .$$

$$r_{20} = [b][bd^+][bd^+B][bd^+Bd]\,\| \to [b][bB]\,\|, \qquad \tau(r_{20}) = B \to dBd .$$

$$r_{21} = [bd^+][bd^+][bd^+B][bd^+Bd]\,\| \to [bd^+][bd^+B]\,\|, \qquad \tau(r_{21}) = B \to dBd .$$

$$r_{22} = [\varepsilon][b][bB]\,\| \to [\varepsilon][S]\,\|, \qquad \tau(r_{22}) = S \to bB .$$

With these actions, the parser is deterministic. That we have indeed obtained a right parser for G_{ab} is seen from the following computations.

$$\$[\varepsilon]\,\|\,ac\$ \overset{r_1}{\Longrightarrow} \$[\varepsilon][a]\,\|\,c\$ \overset{r_3}{\Longrightarrow} \$[\varepsilon][a][ad^*c]\,\|\,\$$$

$$\overset{r_{13}}{\Longrightarrow} \$[\varepsilon][a][aA]\,\|\,\$ \overset{r_{17}}{\Longrightarrow} \$[\varepsilon][S]\,\|\,\$.$$

$$\$[\varepsilon]\,\|\,bc\$ \overset{r_2}{\Longrightarrow} \$[\varepsilon][b]\,\|\,c\$ \overset{r_8}{\Longrightarrow} \$[\varepsilon][b][bd^*c]\,\|\,\$$$

$$\overset{r_{18}}{\Longrightarrow} \$[\varepsilon][b][bB]\,\|\,\$ \overset{r_{22}}{\Longrightarrow} \$[\varepsilon][S]\,\|\,\$.$$

$$\$[\varepsilon]\,|\,ad^ncd^n\$ \overset{r_1}{\Longrightarrow} \$[\varepsilon][a]\,|\,d^ncd^n\$ \overset{r_4}{\Longrightarrow} \$[\varepsilon][a][ad^+]\,|\,d^{n-1}cd^n\$$$

$$\overset{r_6^{n-1}}{\Longrightarrow} \$[\varepsilon][a][ad^+]^n\,|\,cd^n\$ \overset{r_5}{\Longrightarrow} \$[\varepsilon][a][ad^+]^n[ad^*c]\,|\,d^n\$$$

$$\overset{r_{14}}{\Longrightarrow} \$[\varepsilon][a][ad^+]^n[ad^+A]\,|\,d^n\$$$

$$\overset{(r_7r_{16})^{n-1}}{\Longrightarrow} \$[\varepsilon][a][ad^+][ad^+A]\,|\,d\$$$

$$\overset{r_7}{\Longrightarrow} \$[\varepsilon][a][ad^+][ad^+A][ad^+Ad]\,|\,\$$$

$$\overset{r_{15}}{\Longrightarrow} \$[\varepsilon][a][aA]\,|\,\$ \overset{r_{17}}{\Longrightarrow} \$[\varepsilon][S]\,|\,\$, \quad \text{for all } n \geqslant 1 .$$

$$\$[\varepsilon]\,|\,bd^ncd^n\$ \overset{r_2}{\Longrightarrow} \$[\varepsilon][b]\,|\,d^ncd^n\$$$

$$\overset{r_9}{\Longrightarrow} \$[\varepsilon][b][bd^+]\,|\,d^{n-1}cd^n\$ \overset{r_{11}^{n-1}}{\Longrightarrow} \$[\varepsilon][b][bd^+]^n\,|\,cd^n\$$$

$$\overset{r_{10}}{\Longrightarrow} \$[\varepsilon][b][bd^+]^n[bd^*c]\,|\,d^n\$$$

$$\overset{r_{19}}{\Longrightarrow} \$[\varepsilon][b][bd^+]^n[bd^+B]\,|\,d^n\$$$

$$\overset{(r_{12}r_{21})^{n-1}}{\Longrightarrow} \$[\varepsilon][b][bd^+][bd^+B]\,|\,d\$$$

$$\overset{r_{12}}{\Longrightarrow} \$[\varepsilon][b][bd^+][bd^+B][bd^+Bd]\,|\,\$$$

$$\overset{r_{20}}{\Longrightarrow} \$[\varepsilon][b][bB]\,|\,\$ \overset{r_{22}}{\Longrightarrow} \$[\varepsilon][S]\,|\,\$, \qquad \text{for all } n \geqslant 1 .$$

The parses produced are:

$$\tau(r_1r_3r_{13}r_{17}) = (A \to c)(S \to aA) .$$

$$\tau(r_2r_8r_{18}r_{22}) = (B \to c)(S \to bB) .$$

$$\tau(r_1r_4r_6^{n-1}r_5r_{14}(r_7r_{16})^{n-1}r_7r_{15}r_{17}) = (A \to c)(A \to dAd)^n(S \to aA) .$$

$$\tau(r_2r_9r_{11}^{n-1}r_{10}r_{19}(r_{12}r_{21})^{n-1}r_{12}r_{20}r_{22}) = (B \to c)(B \to dBd)^n(S \to bB) .$$

The parser for G_{ab} is an example of an "LR(0) parser". Here "LR" means that the input string is parsed from Left to right and that a Right parse is produced. "0" means that lookahead strings of length zero are used in the reduce actions, that is, there is no lookahead.

The procedure followed above for deriving an LR parser is more or less *ad hoc*, because of the inadequate definition of the equivalence of viable stack strings. Later in this chapter, in Section 6.2, we shall give a definition that yields the equivalence directly, and no additional refinements are needed. To this end, we give in the following a grammatical characterization for the viable stack strings of shift-reduce parsers (and prove some lemmas that will be of use in proving properties of LR parsers).

Let $G = (V, T, P, S)$ be a grammar. String $\gamma \in V^*$ is a *viable prefix* of G if

$$S \underset{\text{rm}}{\Longrightarrow}{}^* \delta A y \underset{\text{rm}}{\Longrightarrow} \delta\alpha\beta y = \gamma\beta y$$

holds in G for some strings $\delta \in V^*$ and $y \in T^*$ and rule $A \to \alpha\beta$ in P. γ is a *complete viable prefix* if here $\beta = \varepsilon$.

First we note:

Fact 6.1 Any viable prefix of grammar G is a prefix of some complete viable prefix of G. □

Most properties of viable prefixes can be derived from the following lemma.

Lemma 6.2 *Let $G = (V, T, P, S)$ be a grammar, π a rule string in P^*, γ, η, and δ strings in V^*, A a nonterminal, and y a string in T^* such that*

(a) $$S \underset{\text{rm}}{\overset{\pi}{\Longrightarrow}} \gamma\eta y = \delta A y \quad \text{in } G, \quad \text{and} \quad \pi \neq \varepsilon \ .$$

In other words, γ is a prefix of some nontrivially derived right sentential form not extending over the last nonterminal. Then there are strings δ' in V^* and y' in T^*, rule strings π' and π'' in P^* and a rule $r = A' \to \alpha'\beta'$ in P such that

(b) $$S \underset{\text{rm}}{\overset{\pi'}{\Longrightarrow}} \delta' A' y' \underset{\text{rm}}{\overset{r}{\Longrightarrow}} \delta'\alpha'\beta' y' = \gamma\beta' y', \quad \beta' y' \underset{\text{rm}}{\overset{\pi''}{\Longrightarrow}} \eta y \ ,$$
$$\pi' r \pi'' = \pi, \quad \textit{and} \quad \alpha' : 1 = \gamma : 1 \ .$$

In other words, derivation (a) contains a segment that proves γ to be a viable prefix, even so that the right-hand side of the rule r "cuts" γ properly.

Proof. The proof is by induction on the length of rule string π. If $|\pi| = 1$, statement (a) implies that $\pi = S \to \gamma\eta y$ is a rule in P. Statement (b) then holds if we choose $\delta' = y' = \varepsilon$, $\pi' = \pi'' = \varepsilon$, $r = \pi$, $\alpha' = \gamma$, and $\beta' = \eta y$. We may thus assume that $|\pi| > 1$ and, as an induction hypothesis, that the lemma holds for the rule strings shorter than π. Statement (a) then implies the existence of strings δ_1 in V^* and y_1 in T^*, a rule string π_1 in P^*, and a rule $r_1 = A_1 \to \omega_1$ such that

(1) $$S \underset{\text{rm}}{\overset{\pi_1}{\Longrightarrow}} \delta_1 A_1 y_1 \underset{\text{rm}}{\overset{r_1}{\Longrightarrow}} \delta_1 \omega_1 y_1 = \gamma\eta y = \delta A y, \quad \text{and} \quad \pi_1 r_1 = \pi \ .$$

Here y_1 must be a suffix of y, that is, $y = xy_1$ for some x. Moreover, either $\gamma = \delta_1\alpha'$ for some $\alpha' \neq \varepsilon$ or $\delta_1 = \gamma\alpha$ for some α. In the former case $\omega_1 = \alpha'\eta x$ and statement (b) holds if we choose $\delta' = \delta_1$, $y' = y_1$, $\pi' = \pi_1$, $\pi'' = \varepsilon$, $r = r_1$, and $\beta' = \eta x$. In the latter case, that is, when $\delta_1 = \gamma\alpha$, we may write the first derivation segment in (1) as

(2) $$S \underset{\text{rm}}{\overset{\pi_1}{\Longrightarrow}} \gamma\eta_1 y_1 = \delta_1 A_1 y_1 \ ,$$

where η_1 denotes αA_1. As here $\pi_1 \neq \varepsilon$, we can apply the induction hypothesis to $\pi_1, \gamma, \eta_1, \delta_1, A_1$, and y_1, and conclude that there are strings δ' in V^* and y' in T^*, rule strings π' and π_2, and a rule $r' = A' \to \alpha'\beta'$ such that

$$
\begin{aligned}
&S \underset{\text{rm}}{\overset{\pi'}{\Longrightarrow}} \delta' A' y' \underset{\text{rm}}{\overset{r'}{\Longrightarrow}} \delta'\alpha'\beta' y' = \gamma\beta' y', \beta' y' \underset{\text{rm}}{\overset{\pi_2}{\Longrightarrow}} \eta_1 y_1 \ , \\
&\pi' r' \pi_2 = \pi_1, \quad \text{and} \quad \alpha' : 1 = \gamma : 1 \ .
\end{aligned} \tag{3}
$$

Here we have

$$
\eta_1 y_1 = \alpha A_1 y_1 \underset{\text{rm}}{\overset{r_1}{\Longrightarrow}} \alpha\omega_1 y_1 = \eta y \ . \tag{4}
$$

Recall that $\delta_1 = \gamma\alpha$ and $y = xy_1$ in (1), and so $\alpha\omega_1 = \eta x$. By combining (3) and (4) and choosing $\pi'' = \pi_2 r_1$ we can conclude that statement (b) holds. □

As an immediate consequence of Lemma 6.2 we have:

Lemma 6.3 *Let $G = (V, T, P, S)$ be a grammar, δ a string in V^*, y a string in T^*, and A a nonterminal such that*

$$
S \underset{\text{rm}}{\Longrightarrow}^{+} \delta A y \qquad \text{in } G \ .
$$

Then δA is a viable prefix of G.

Proof. Choose $\gamma = \delta A$ and $\eta = \varepsilon$ in Lemma 6.2. □

As an other application of Lemma 6.2 we prove the following important result.

Lemma 6.4 *Any prefix of a viable prefix is a viable prefix.*

Proof. Let γ_1 and γ_2 be strings such that $\gamma_1\gamma_2$ is a viable prefix. We prove that γ_1 is a viable prefix. By definition,

$$
S \underset{\text{rm}}{\Longrightarrow}^{n} \delta A y \underset{\text{rm}}{\Longrightarrow} \delta\alpha\beta y = \gamma_1\gamma_2\beta y \tag{1}
$$

for some $n \geq 0$, string δ, terminal string y, and rule $A \to \alpha\beta$. Here δ is a prefix of γ_1, or $\delta \neq \varepsilon$ and γ_1 is a prefix of δ. In the former case, derivation (1) proves γ_1 as a viable prefix because we may write $\alpha\beta$ as $\alpha'\beta'$, where $\delta\alpha' = \gamma_1$ and $\beta' = \gamma_2\beta$. In the latter case, we may write $\delta A y$ as $\gamma_1\eta y$ for some η. Because $\delta \neq \varepsilon$ implies $n > 0$, we can then conclude by Lemma 6.2 that γ_1 is a viable prefix. □

The following lemma states how viable prefixes rightmost derive viable prefixes.

Lemma 6.5 *Let $G = (V, T, P, S)$ be a reduced grammar, γ a string in V^*, and $A \to \alpha\beta$ a rule in P. If γA is a viable prefix of G, then so is $\gamma\alpha$.*

此为试读,需要完整PDF请访问: www.ertongbook.com