

Course: Compiler Construction

Week 7: Bottom-Up Parsing

Lecturer: Martha Gichuki

Learning outcomes Week 7: Bottom-Up Parsing

At the end of the lecture, you will be able to:

- (i) Define Shift Reduce Parsing, Left Right Parser
- (ii) Implement Shift Reduction
- (iii) Describe stack implementation in Bottom-up Parsing

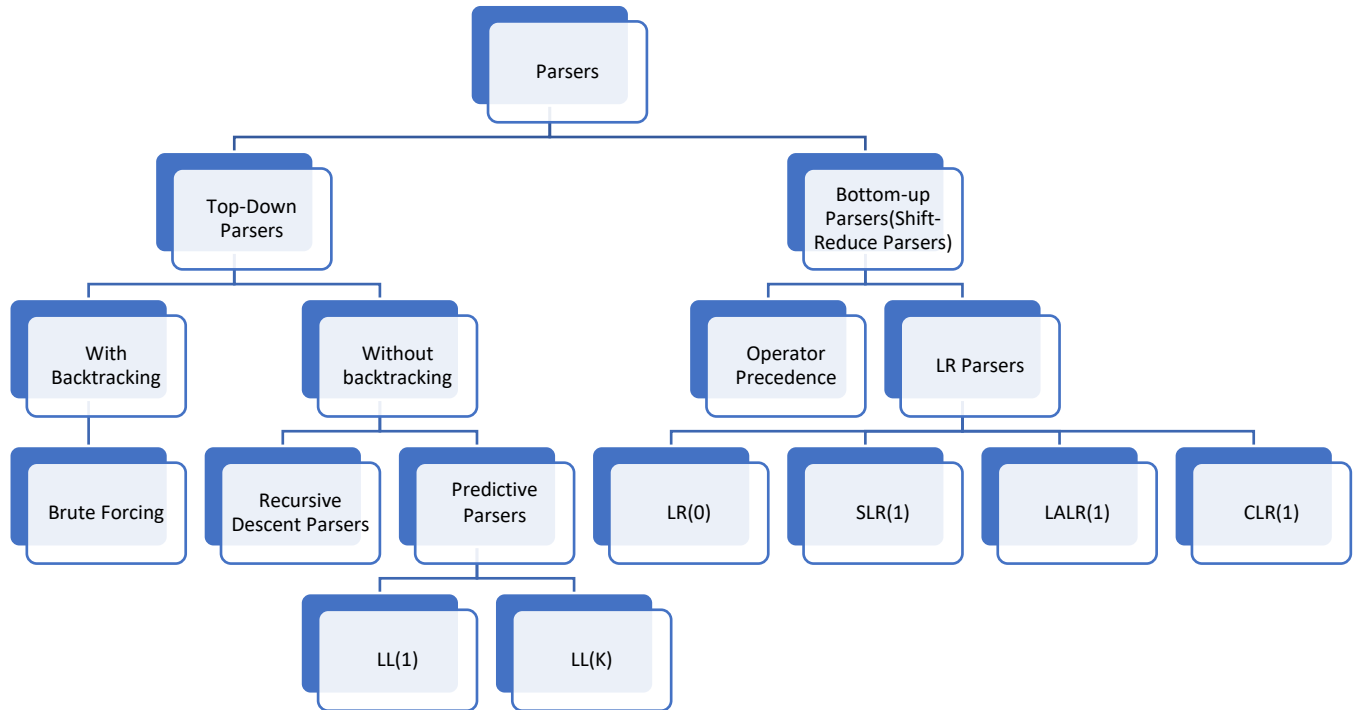
1. Introduction

1.1. Definition: Parsing

- ✓ Parsing is a convenient way to see how strings are derived from the start symbol.
- ✓ It is a way of analyzing a string or text into logical syntactic components
- ✓ Tokens are divided into parts and the relations between the parts can then be described¹

¹ Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 86

1.2. Classification of parsers²



1.3. BOTTOM-UP PARSING

- ✓ This approach is also known as **shift-reduce parsing**
- ✓ It is the primary parsing method for many compilers, due to its **speed** and the **tools** which automatically generate a parser based on the grammar³.
- ✓ Generally, bottom-up parsing **starts from the leaf nodes of a tree and works in upward direction till it reaches the root node.**
- ✓ This parsing strategy is based on the **reverse process to top-down parsing.**

² Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 324

³ Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 322

Example:- Consider the grammar below:

$$1 \quad S \rightarrow aABe$$

$$2 \quad A \rightarrow Abc|b$$

$$3 \quad B \rightarrow d$$

To parse the sentence **abcde** using the bottom-up approach gives the following reductions:-

abcde

aABCDE by 2

aAde by 2

aABe by 3

S by 1

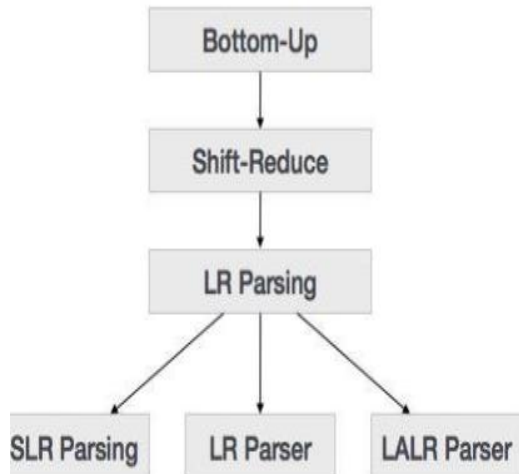
Reverse gives in right most derivation:

$$S \rightarrow aABe \rightarrow aAde \rightarrow aAbcde \rightarrow abcde$$

- ✓ Here, we start from a sentence and then apply production rules in **reverse manner in order to reach the start symbol**.
- ✓ Instead of expanding successful non-terminals according to production rules, a current string or right sentential form is collapsed each time until the start non-terminal is reached to predict the legal next symbol; i.e. it can be regarded as a **series of reductions**⁴.
- ✓ The figure below is a summary of the bottom-up parsers available⁵

⁴ Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 27

⁵ Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 323



2. LR Parsing

- ✓ The LR parser is a non-recursive, shift-reduce, bottom-up parser.
- ✓ It uses a wide class of context-free grammar which makes it the most efficient syntax analysis technique.
- ✓ It is one of the best methods of syntactic recognition of programming languages⁶.
- ✓ The **L** stands for **left to right** scan, and **R** stands for **right most derivation in reverse**.
- ✓ In general, we can have ***LR(k)* parsing with *k* symbols of LOOKAHEAD**.
- ✓ However, **LR parsing** refers to ***LR(1)* parsing**.

Advantages of LR Parsing⁷

- i) LR parsers can recognize virtually **all programming language constructs** written with CFG grammars.
- ii) It is the **most general, non-backtracking** technique known.
- iii) It can be implemented in a **very efficient** manner.

⁶ Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 324

⁷ Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 266

iv)The language it can recognize is a proper **super set of that of predictive parsers.**

v)It can recognize **syntax errors quickly.**

Disadvantages of LR parsing⁸

- ✓ The primary drawback of LR parsers is that they **require too much work to manually create LR parsing tables.**
- ✓ However, **tools exist to generate LR parsers** from a given grammar i.e. parser generators such as Yet Another Compiler Compiler (YACC), BISON etc.

3. LR Parsing Methods

- ✓ There are three widely used algorithms available for constructing an LR parser:

i. Simple LR (SLR)

- ✓ **Easy to implement but less powerful** than other parsing methods.
- ✓ Generally, works on **smallest class of grammar and have a few numbers of states, hence a very small table⁹**

ii. Canonical LR

- ✓ This is the most general and most powerful.
- ✓ However, it is **tedious and costly to implement**, i.e. for the same grammar, it has got much number of states as compared to SLR parsers.

⁸ Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 267

⁹ Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 325

- ✓ Generally, works on complete set of LR(1) Grammar and generates a larger table and more states¹⁰

iii. LOOKAHEAD LR (LALR)

- ✓ A mixture of SLR and canonical LR, but it can be implemented efficiently i.e. it contains the same number of states as Simple LR parser for the same grammar.
- ✓ Notice that most parser generators generate LALR parsers since they are a trade-off between power and efficiency¹¹.

3.1. Implementing Shift Reduction

- ✓ A shift-reduce parser is implemented using the following notation¹²:

An input stream, containing

a phrase to be parsed and

a stack holding a symbols

- ✓ The **input stream** holds terminals, the stack can hold a mixture of terminals and non-terminals, the latter generated by earlier reductions.
- ✓ The operation **shift** moves a symbol from the input to the stack while the operation **reduce** combines the sequence ending with the last terminal shifted to form a non-terminal on the stack.
- ✓ When the input is exhausted, the single start symbol should be presented assuming all reductions have been performed¹³.

¹⁰ Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 325

¹¹ Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 324

¹² Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 281

¹³ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 279

✓ **Example:- Consider the following grammar¹⁴:**

$exp \rightarrow exp + exp$

$exp \rightarrow exp * exp$

$exp \rightarrow (exp)$

$exp \rightarrow id$

Example:- Show the sequence of shift/reduce for the string $id_1 + id_2 * id_3$.¹⁵

Stack	Input	Action
\$	id1+id2*id3\$	shift
\$id1	+id2*id3\$	reduce using $exp \rightarrow id$
\$exp	+id2*id3\$	shift
\$exp+	id2*id3\$	shift
\$exp+id2	*id3\$	Reduce using $exp \rightarrow id$
\$exp+exp	*id3\$	shift
\$exp+exp*	id3\$	shift
\$exp+exp*id3	\$	Reduce using $exp \rightarrow id$
\$exp+exp*exp	\$	Reduce using $exp \rightarrow exp * exp$
\$exp+exp	\$	Reduce using $exp \rightarrow exp + exp$
\$exp	\$	Accept

¹⁴ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 279

¹⁵ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 259

Comparison between LL and LR¹⁶

LL	LR
Does a left-most derivation	Does a right-most derivation in reverse
Starts with the root non-terminal on the stack	Ends with the root non-terminal on the stack
Ends when the stack is empty	Starts with an empty stack
Uses the stack for designating what is still to be expected	Uses the stack for designating what is already seen
Builds the parse tree top-down	Builds the parse tree bottom -up
Continuously pops a non-terminal off the stack and pushes the corresponding right-hand side	Tries to recognize a right-hand side on the stack, pops it and pushes the corresponding non-terminal
Expands the non-terminals	Reduces the non-terminals
Reads the terminals when it pops one off the stack	Reads the terminals while it pushes them on the stack
Pre-order traversal of the parse tree	Post-order traversal of the parse tree

Handles

- ✓ A handle of a right-sentential form γ is a production $A \rightarrow \beta$ and a position in γ where β may be found and replaced by A to produce the previous right sentential form in a **rightmost derivation** of γ

¹⁶ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 27

- ✓ This means that if $S \rightarrow *r\alpha A w \rightarrow r\alpha\beta w$ then $A \rightarrow \beta$ in the position following α is a handle of $\alpha\beta w$ ¹⁷
- ✓ Because γ is a **right-sentential form**, the substring to the right of a handle contains **only terminal symbols**
- ✓ This means that a handle is a string that can be reduced and also allows further reductions back to the start symbol (using a particular production at a specific spot)¹⁸.
- ✓ Informally, a "**handle**" is a substring that **matches the body of a production**, and whose reduction represents one step along the reverse of a rightmost derivation.
- ✓ **Note:-** The string w to the right of the handle must only contain terminal symbols.
- ✓ For convenience, we refer to the **body β rather than $A \rightarrow \beta$ as a handle**¹⁹.

3.2. Handle-pruning

- ✓ The process to construct a bottom-up parse is called **handle-pruning** that is start with a **string of terminals w** to be parsed.
- ✓ If w is a sentence of the grammar at hand, then let $w = \gamma_n$, where γ_n is the n^{th} **right-sentential form** of some; yet **unknown** rightmost derivation

$$S = \gamma_0 \rightarrow \gamma_1 \rightarrow \gamma_2 \rightarrow \dots \rightarrow \gamma_n = w$$
²⁰
- ✓ To reconstruct this derivation in **reverse order**, we **locate the handle β_n** in γ_n and replace β_n by the **head** of the relevant production $A_n \rightarrow \beta_n$ to obtain the previous right-sentential form γ_{n-1} .
- ✓ Repeat this process. i.e. locate the handle **β_{n-1} in γ_{n-1}** and reduce it to obtain the right-sentential form γ_{n-2} .

¹⁷ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 27

¹⁸ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 27

¹⁹ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 27

²⁰ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 260

- ✓ At the end of this process, if we produce a **right-sentential form** with only the **start symbol S**, then we halt and announce successful completion of parsing²¹.

Algorithm

Set **i to n** and apply the following simple algorithm

For i = n down to 1

1. Find the handle $A_i \rightarrow \beta_i$ in γ_i
2. Replace β_i with A_i to generate γ_{i-1}

- ✓ This takes 2^n steps,
- ✓ Where **n** is the **length of the derivation**²²

4. Stack implementation

- ✓ One scheme to implement a handle-pruning, bottom-up parser is to use a **shift-reduce parser**.
- ✓ Shift-reduce parsers use a **stack** and an input buffer as shown below²³:-

1. Initialize stack with **\$**
2. Repeat until the top of the stack is the goal symbol and the **input token** is **\$**
 - i). find the handle if we don't have a handle on top of the stack, shift an input symbol onto the stack
 - ii). prune the handle if we have a handle $A \rightarrow \beta$ on the stack, reduce
 - a) pop $|\beta|$ symbols off the stack

²¹ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 260

²² Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 261. 273

²³ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 262

b) push **A** onto the stack²⁴

Generally, a shift-reduce parser has four canonical actions:

1. **shift** - next input symbol is shifted onto the top of the stack
2. **reduce** - right end of handle is on top of stack;
 - ✓ locate left end of handle within the stack;
 - ✓ pop handle off stack and push appropriate nonterminal LHS
3. **accept** - terminate parsing and signal success
4. **error** - call an error recovery routine²⁵

LR Parsers

- ✓ As with LL(1), the aim of LR parsers is to **make the choice of action** depend only on the next input symbol and the symbol on top of the stack.
- ✓ To achieve this, a DFA needs to be **constructed**.
- ✓ The DFA is used to determine the next action and it only needs to look at the **current state** (stored at the top of the stack) and the **next input symbol** (*shift action*) or **non-terminal** (*reduce action*).
- ✓ We represent the DFA as a table, where we cross-index a DFA state with a symbol (terminal or non-terminal)²⁶.
- ✓ Generally, we need to encode the DFA in a table i.e. a shift-reduce parser's DFA can be encoded in two tables
- ✓ One row for each state

²⁴ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 262

²⁵ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 184

²⁶ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 184

- I. **action table** encodes what to do given the current state and the next input symbol
- II. **goto table** encodes the transitions to take after a reduction²⁷

Once the DFA has been encoded as a table, one of the following actions should be performed:-

Actions (1)

Given the current state and input symbol, the main possible actions are:-

- ✓ **si** – shift the input symbol and state **i** onto the stack (i.e., shift and move to **state i**)
- ✓ **rj** – reduce using grammar **production j**
- ✓ The production number tells us how many <symbol, state> pairs to **pop** off the stack²⁸

Actions (2)

Other possible action table entries

- **accept**
- **blank** – no transition, which means syntax error
- ✓ A LR parser will **detect an error** as soon as possible on a left-to-right scan
- ✓ real compiler needs to produce an **error message, recover, and continue parsing** when this happens²⁹

²⁷ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 184

²⁸ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 185

²⁹ Compilers: Principles, Techniques and Tools Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 185

Content Covered in Week 7: Bottom-up Parsing

At the end of the lecture, we were able to:

- i. Define Shift Reduce Parsing, Left Right Parser
- ii. Implement Shift Reduction
- iii. Describe stack implementation in Bottom-up Parsing

Course Text Books

1. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007))
2. Compiler Construction: Principles and practices; Kenneth C. Louden; Cengage Learning; 1st edition, ISBN-10: 0534939724 (1997)
3. Basics of Compiler Design: Torben Mogensen; DIKU University of Copenhagen Universitetsparken 1 DK-2100 Copenhagen DENMARK (2007)
4. Engineering a Compiler: 2nd edition; Keith D. Cooper and Linda Torczon; Morgan Kaufmann Publishers ISBN: 978-0-12-088478-0 (2003)
5. Compiler Design: Santanu Chattopadhyay; PHI Learning Publishers, ISBN 812032725X, 9788120327252 (2005)