Bottom Up Parser

Dr. Meera Thapar Khanna

CSE Department

Pandit Deendayal Energy University

BOTTOM-UP PARSING

Bottom-up parsing corresponds to the construction of a parse tree for an input string beginning at the leaves (the bottom nodes) and working up towards the root (the top node).

It involves —reducing an input string ‘w’ to the Start Symbol of the grammar.

For each reduction step, a particular substring matching the right side of the production is replaced by symbol on the left of that production and it is the Right most derivation.

Example

Grammar

S → aABe

A → Abc | b

B → d

Input string is
‘abbcde’

Reduction (Left)

a**b**bcde	A -> b
a**Abc**de	A -> Abc
aA**d**e	B -> d
aABe	S -> aABe

Derivation (Right)

S -> aABe

-> aAde

-> aAbcde

-> abbcde

Example

Grammar:

E -> E+T|T

T -> T*F|F

F -> (E)|id

Input string

id*id

Reduction (Left)	
Id*id	F -> id
F*id	T -> F
T***id**	F -> id
T*F	T -> T*F
T	E -> T

Derivation (Right)

E -> T

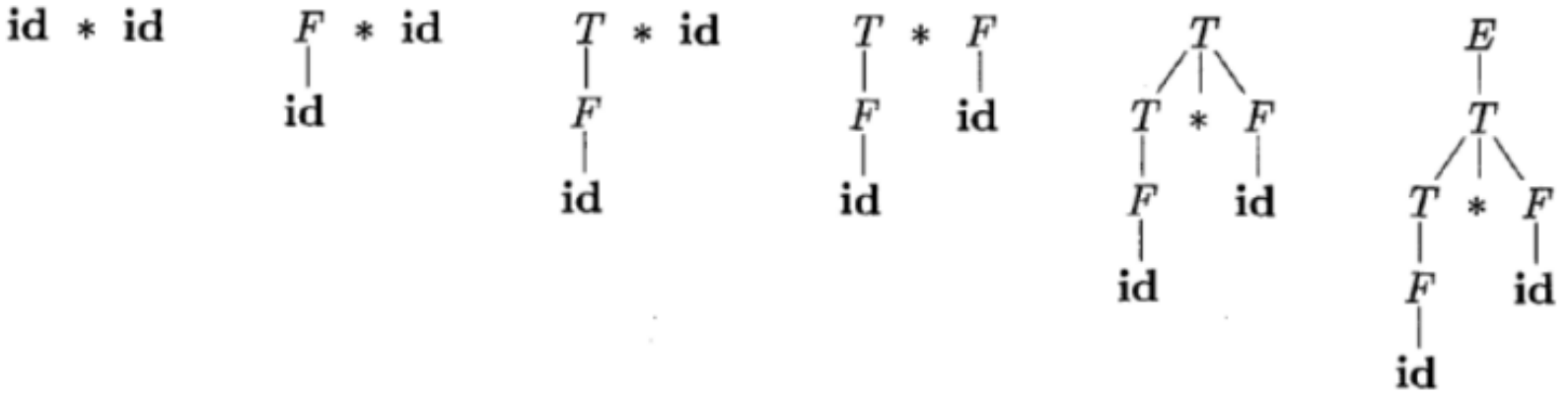
-> T*F

-> T*id

-> F*id

-> id*id

A Bottom-up Parse tree for the input String "id*id"



Handle

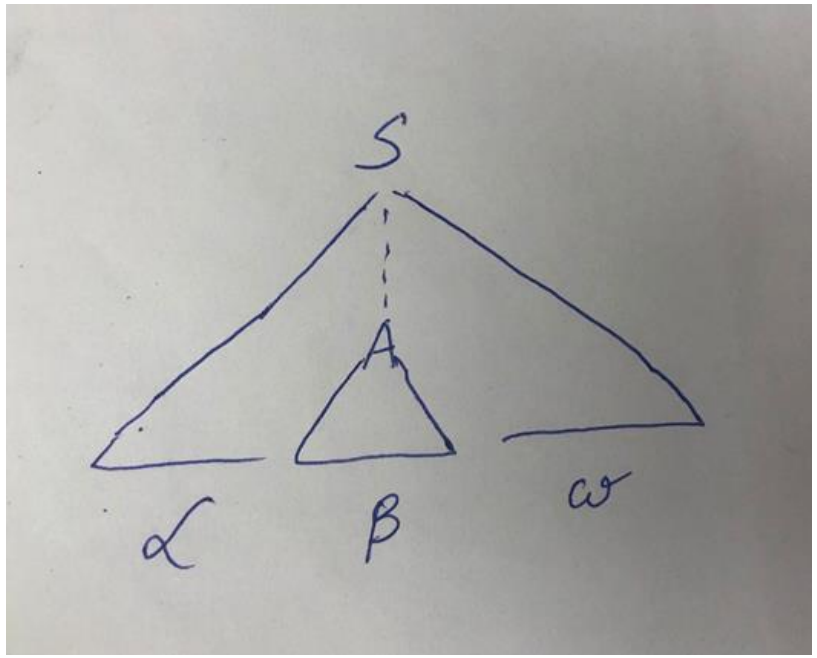
- A handle of a string is a substring that matches the right side of a production rule and reduction to one step in the reverse of a rightmost derivation.
- For instance A substring α of the tree's that matches some production A → α where reducing α to A is one step in the reverse of a rightmost derivation.

Contd.

A handle of a right-sentential form γ is a production A → β 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.

- if S ⇒rm* αAw ⇒rm αβw then A → β in the position following α is a handle of αβw
- Because γ is a right-sentential form, the substring to the right of a handle contains only terminal symbols.

If G is unambiguous then every right-sentential form has a unique handle



A handle A -> β in the parse tree for αβw

Handle Pruning

The process to construct a bottom-up parse is called handle-pruning. To construct a rightmost derivation

$$S = \gamma 0 \Rightarrow \gamma 1 \Rightarrow \gamma 2 \Rightarrow \cdots \Rightarrow \gamma n-1 \Rightarrow \gamma n = w$$

The following algorithm is used:

for i = n to 1

find the handle $A_i \rightarrow \beta_i$ in γ_i

replace β_i with A_i to generate γ_{i-1}

This takes 2n steps, where n is the length of the derivation

SHIFT-REDUCE PARSING

One scheme to implement a handle-pruning, bottom-up parser is called a shift-reduce parser.

Shift-reduce parsers use a stack and an input buffer

1. initialize stack with $ (used to mark bottom of the stack/ right end of the input)
2. Repeat until the top of the stack is the desired symbol and the input token is $

Find the handle

if we don't have a handle on top of the stack, shift an input symbol onto the stack

Prune the handle

if we have a handle $A \rightarrow \beta$ on the stack, reduce

i) pop | β | symbols off the stack

ii) push A onto the stack

Actions in Shift Reduce Parser

- Shift: Next input symbol is shifted onto the top of the stack.
- Reduce: The parser replaces the handle within the stack with a non-terminal.
- Accept: Successful Completion.
- Error: Discover error and call error recovery method.

Shift-Reduce Parser Stack Implementation

STACK	INPUT	ACTION
$	Id*id$	Shift
$id	*id$	Reduce with F -> d
$F	*id$	Reduce with T -> F
$T	*id$	Shift
$T*	id$	Shift
$T*id	$	Reduce with F -> id
$T*F	$	Reduce with T -> T*F
$T	$	Reduce with E -> T
$E	$	Accept

Conflicts in Shift Reduce Parsing

1. Shift Reduce Conflict: The parser cannot decide whether to shift or reduce.
2. Reduce Reduce Conflict: The parser cannot decide which of several reductions to make.

Example

Grammar:

S -> aAcBe

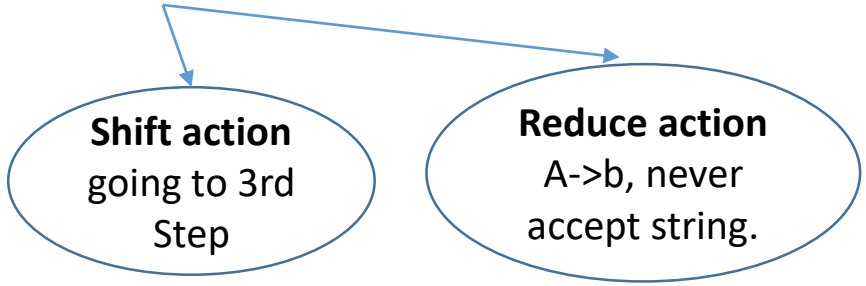
A -> Ab|b

B -> d

Input string is —”abbcde”.

The series of shift and reductions to the start symbol are as follows.

- abbcde -> aAbcde -> aAcde -> aAcBe -> S



Example

Grammar

E -> E+E | E*E | id

Input: id+id*id

Shift-Reduce Conflict

STACK	INPUT	ACTION	STACK	INPUT	ACTION
$	Id+id*id$	Shift	$	Id+id*id$	Shift
$id	+id*id$	Reduce with E -> id	$id	+id*id$	Reduce with E -> id
$E	+id*id$	Shift	$E	+id*id$	Shift
$E+	Id*id$	Shift	$E+	Id*id$	Shift
$E+id	*id$	Reduce with E -> id	$E+id	*id$	Reduce with E -> id
$E+E	*id$	Reduce with E -> E+E	$E+E	*id$	Shift
$E	*id$	Shift	$E+E*	id$	Shift
$E*	id$	Shift	$E+E*id	$	Reduce with E -> id
$E*id	$	Reduce with E -> id	$E+E*E	$	Reduce with E -> E*E
$E*E	$	Reduce with E -> E*E	$E+E	$	Reduce with E -> E+E
$E			$E		

Example

Grammar

R -> R+R | R+c | R

R -> c

Input: c+c

Reduce-Reduce Conflict

STACK	INPUT	ACTION	STACK	INPUT	ACTION
$	c+c$	Shift	$	c+c$	Shift
$c	+c$	Reduce with R -> c	$c	+c$	Reduce with R -> c
$R	+c$	Shift	$R	+c$	Shift
$R+	c$	Shift	$R+	c$	Shift
$R+c	$	Reduce with R -> c	$R+c	$	Reduce with M -> R+c
$R+R	$	Reduce with M -> R+R	$M	$	
$M	$				