Bottom up parsers

Dr g sudha Sadasivam

Shift Reduce parsing

- Constructs a parse tree for an input string beginning at the leaves(the bottom) and working up towards the root(the top)
- It is “reducing” a string w to the start symbol of a grammar
- Bottom-up parsing is also known as *shift-reduce parsing*

- **Shift step**:
  - The shift step refers to the advancement of the input pointer to the next input symbol, which is called the shifted symbol.
  - This symbol is pushed onto the stack.
  - The shifted symbol is treated as a single node of the parse tree.
- **Reduce step** :
  - When the parser finds a complete grammar rule (RHS) and replaces it to (LHS), it is known as reduce-step.
  - This occurs when the top of the stack contains a handle.
  - To reduce, a POP function is performed on the stack which pops off the handle and replaces it with LHS non-terminal symbol.

- If the substring is chosen correctly, the right most derivation of that string is created in the reverse order.

  Rightmost Derivation: $S \Rightarrow \omega$

  Shift-Reduce Parser finds: $\omega \Leftarrow \ldots \Leftarrow S$

- Consider the grammar

  S → aABe

  A → Abc | b

  B → d

Input string :

abbcde

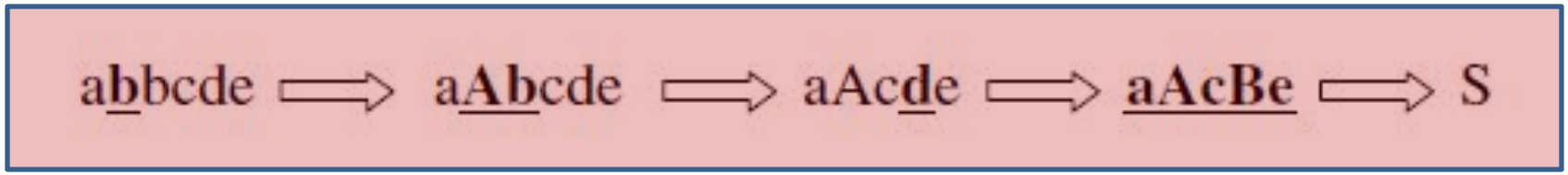
aAbcde

aAde

⇓ reduction

aABe

S



- Rightmost derivation in reverse
- Bottomup parsing involves finding handles and reducing them
- A substring which is the right hand side of the production such that the replacement of that substring by the production left hand side leading eventually to the reduction to the start symbol, by the reverse of rightmost derivation is called a "**handle**".

  A□b at position 2; A□Ab at position 2; B□d at position 4; S□aAcBe at position 1

- A rightmost derivation in reverse called canonical reduction sequence is obtained by the process of **handle pruning** (identify handle and reducing).

Actions of shift-reduce parser

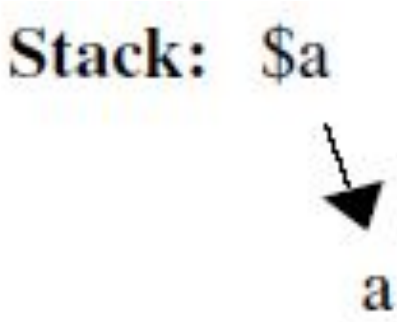
1. **Shift Action:** Here the next input symbol is shifted on to the top of the stack.
2. **Reduce Action:** The parser knows that the right end of the handle is on the top of the stack. It locates the left end of the handle and replaces the handle by the left side of the production.
3. **Accept Action:** The parser announces the successful completion of parsing when $ S is on the top of the stack and $ is at the input.
4. **Error Action:** The parser discovers that a syntax error has occurred and calls the error recovery routine.

S → aAcBe

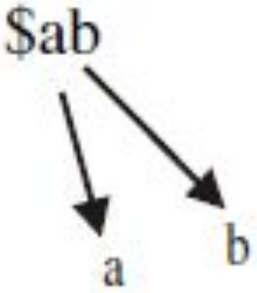
A → Ab | b

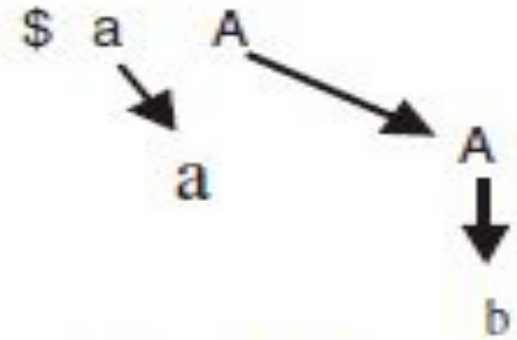
B → d

Stack	Input buffer	Production
$	abbcde$	Shift
$a	bbcde$	Shift
$ab	bcde$	Reduce using A→ b
$aA	bcde$	Shift
$aAb	cde$	Reduce using A → Ab
$aA	cde$	Shift
$aAc	de$	Shift
$aAcd	e$	Reduce using B→d
$aAcB	e$	Shift
$aAcBe	$	Reduce using S → aAcBe
$S	$	Accept.

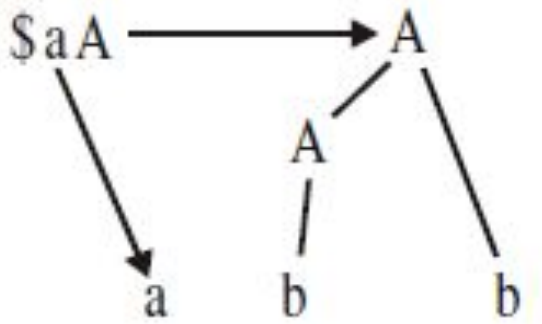


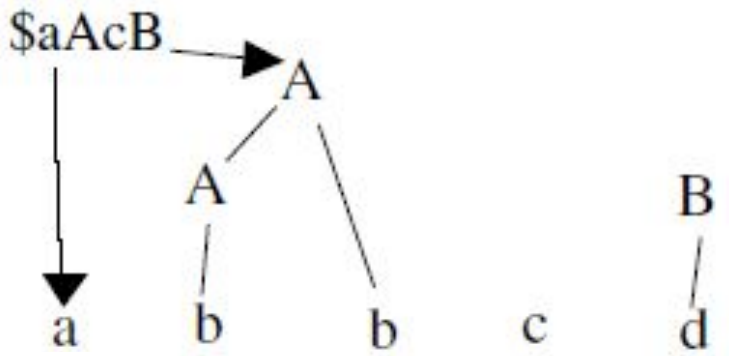
Shift – create a new node

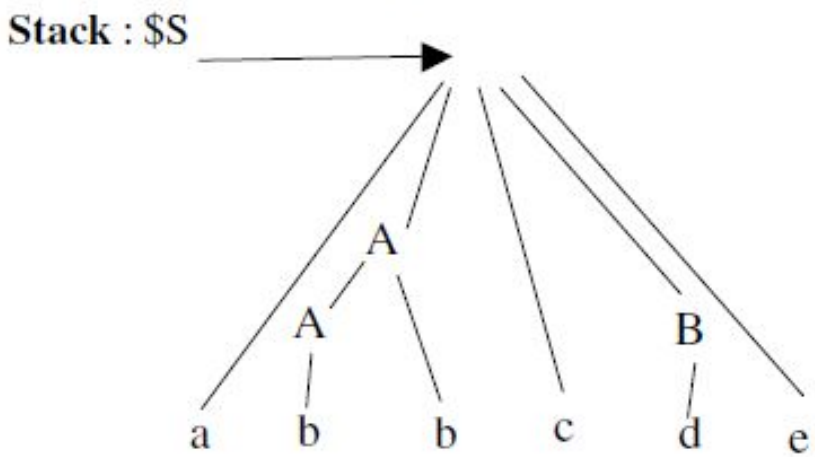




Reduce- create a new node and make it as parent







Example

- S□a | ^ | (T)
- T□ T, S | S

Find rightmost derivation for (a,(a,a))

Show actions of shift reduce parser

S → (L) | a

L → L , S | S

Parse the input string

(a , (a , a))

Stack	Input Buffer	Parsing Action
$	(a,(a,a))$	Shift
$(	a,(a,a))$	Shift
$(a	,(a,a))$	Reduce S → a
$(S	,(a,a))$	Reduce L → S
$(L	,(a,a))$	Shift
$(L,	(a,a))$	Shift
$(L,(	a,a))$	Shift
$(L,(a	,a))$	Reduce S → a
$(L,(S	,a))$	Reduce L → S
$(L,(L	,a))$	Shift
$(L,(L,	a))$	Shift
$(L,(L,a	))$	Reduce S → a
$(L,(L,S)	))$	Reduce L → L , S
$(L,(L	))$	Shift
$(L,(L)	)$	Reduce S → (L)
$(L,S	)$	Reduce L → L , S
$(L	)$	Shift
$(L)	$	Reduce S → (L)
$S	$	Accept

Considering the string “10201”, design a shift-reduce parser for the following grammar-

S → 0S0 | 1S1 | 2

Stack	Input Buffer	Parsing Action
$	1 0 2 0 1 $	Shift
$ 1	0 2 0 1 $	Shift
$ 1 0	2 0 1 $	Shift
$ 1 0 2	0 1 $	Reduce S → 2
$ 1 0 S	0 1 $	Shift
$ 1 0 S 0	1 $	Reduce S → 0 S 0
$ 1 S	1 $	Shift
$ 1 S 1	$	Reduce S → 1 S 1
$ S	$	Accept

Operator Precedence Parser

1. There are no ϵ productions in this grammar.
2. No production would have two adjacent non-terminals.

Example of operator grammar is

- E → E + E | E - E | id | (E).

The precedence relationships between the operators a and b are

1. The relation a <· b implies that the terminal 'a' has lower precedence than 'b'.
2. The relation a ·> b implies that the terminal 'a' has higher precedence than 'b'.
3. The relation a = b implies that the terminal 'a' has same precedence as 'b'.

Parsing Action

1. Scan from left to right until the right most ·> is encountered.
2. Scan towards left over all equal precedence until the first <· precedence is encountered
3. Everything between <· and ·> is a handle.
4. Once the handle is identified reduce it.

```
if $S is on Top Of Stack (TOS) and $ is in input then accept
Else  {
      if 'a' is in TOS and 'b' is in input.
            if a <· b or a =  b
                Then shift 'b' onto TOS
                 Else if a ·> b
                        then
                              pop TOS until input <· TOS  and reduce
                       else  error
 }
```

	a	↑	(	)	,	$
a				•>	•>	•>
↑				•>	•>	•>
(	<•	<•	<•	=	<•	
)				•>	•>	•>
,	<•	<•	<•	•>	•>	
$	<•	<•	<•			

S → (L) | a

L → L , S | S

Parse the input string

(a , a)

Stack	Input	Operation
$	(a,a)$	$ <• (, shift
$(	a,a)$	(<• a, shift
$(a	,a)$	a •> , ; reduce using S
$(S	,a)$	(<• , shift
$(S,	a)$	, <• a , shift
$(S,a	)$	a •>) reduce using S
$(S,S	)$	, •>) reduce using T
$(S,T	)$	,•>)redusing using T
$(T)	$	) •> $ reduce using S
$S		