Saturday
30-05-2020

CD Assignment-2

UNIT-2

N.V.S.K.kalyani
174N1A0584
3rd B.Tech CSE-C

2 Marks

1. what are the error recover strategies of a parser?

Ans Few error Recovery Strategies are:

→ panic mode recovery
→ pharse level recovery
→ Error-productions
→ Global correction

2. write a code for parser generator of if-else statement?

Ans (lex program: if t-I)

```
alpha [A-Za-z]
digit [0-9]
%.%
[\t \n]
if return zf;
then return THEN;
else return ELSE;
{digit}+ return Num;
{alpha}({alpha}/{digit}) * return ID;
"<=" return LE;
">=" return GE;
"==" return EQ;
"!=" return NE;
"||" return OR;
"&&" return AND;
return utex[0];
%.%
```

3. outline about derivation tree with an Example?

Ans Derivation tree the root is the start variable all internal nodes are labelled with variables while all leaves are labelled with terminals. The children of an internal node are labelled from left to right with the right-hand side of the production used.

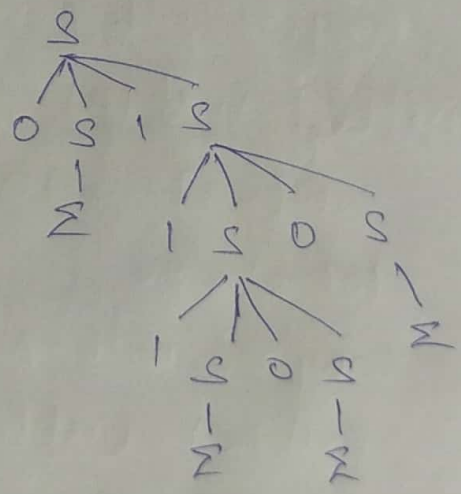
Ex: $S \rightarrow 0S1S \mid 1S0S \mid \Sigma$

The derivation for 011100 given earlier was left most:

$S \Rightarrow 0S1S \Rightarrow 01S \Rightarrow 011S0S \Rightarrow 0111S0S\Sigma0S$

$\Rightarrow 01110S0 \Rightarrow 011100S \Rightarrow 011100$

tree:




4. remove left factoring for the CFG

$S \rightarrow ietS \mid ietSeS \mid a$

$\epsilon \rightarrow b$

Ans $S \rightarrow ietS \mid ietSeS \mid a$

compose $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$

then $S \rightarrow ietSS'$

$S' \rightarrow \epsilon S \mid \Sigma$

∴ CFG are $S \rightarrow iEtSS' | a$

$S' \rightarrow eS | \epsilon$

$E \rightarrow b$

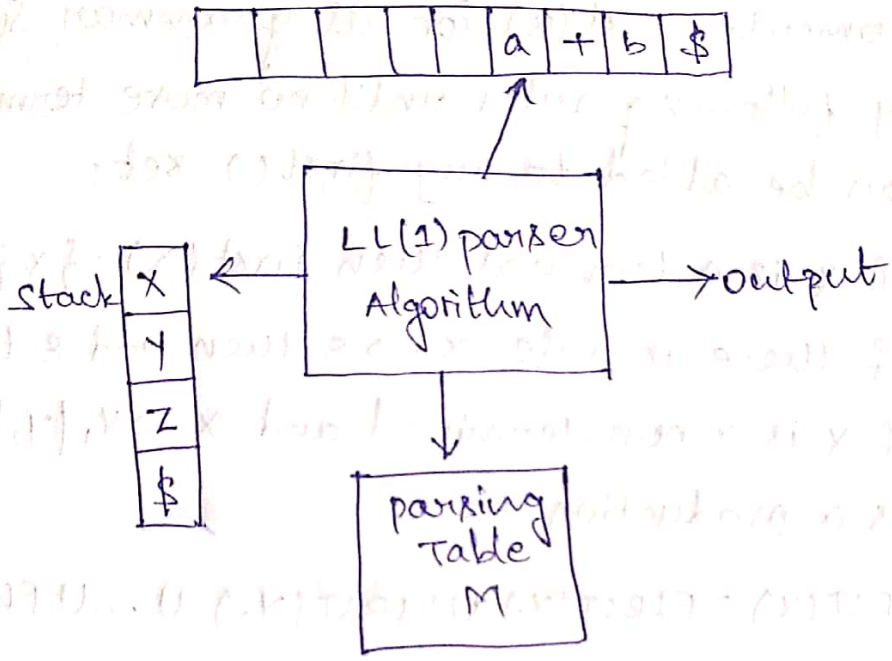
5. Discuss Backtracking?

Ans Backtracking is a technique in which for expan-sion of non terminal symbol we choose are production rule if mismatch occur then we will try with another alternative any for derivation.

20)

# LL(1) parser

## Introduction:-




- L - It reads input string from left to right.
- L - It uses leftmost derivation for i/p string.
- (1) - In the i/p symbol means it uses only one i/p symbol.

## Construction of LL(1) parser:-

For construction of predictive LL(1) parser we have the following steps:

1) Compute First() and Follow() functions for the given CFG production Rules.
2) Construct the predictive parsing Table using First and Follow functions.
3) Parse the input string with the help of LL(1) parsing Algorithm that uses input buffer, stack and parsing Table.

Algorithm:-

set ip point to the first symbol of w$;
set x to the top stack symbol
let a as current i/p symbol;
while (x != $) {
if (x is a terminal & equals to i/p symbol)
pop the stack and advance ip;
else if (x is a terminal)
error();
else if (M[x,a] is an error entry)
error();
else if (M[x,a] = x → Y1, Y2, ... Yk) {
output the productions x → Y1, Y2, ... Yk
pop the stack;
push Yk, Y2, Y1 on the stack with Y1 on top
}
set x to the top stack symbol;
}
if (x == $ and a == $)
write "success";

2b) Construct LL(1) parsing table for the grammar

S → iEtSS' | a
S' → eS | ε
E → b

| Non terminal | Input symbol | | | | | |
|---|---|---|---|---|---|---|
| | a | b | e | i | t | $ |
| S | S→a | error | error | S→ietss' | error | error |
| S' | error | error | S'→es | error | error | S'→ϵ |
| ϵ | error | ϵ→b | error | error | error | error |

let S→ietss'

compare with A→α

First[α] = first(ietss') = {i}

∴ M[S,i] = S→ietss'

consider S→a

compare with A→α

First[α] = first(a) = {a}

∴ M[S,a] = S→a

let S'→es

compare with A→α

first(α) = First(es) = {e}

∴ M[S',e] = S'→es

let S'→Σ

compare with A→α

first(α) = first(Σ) = {Σ}

follow(S') = follow(S) = {$}

∴ M[S',$] = S'→ϵ

let ϵ→b

compare with A→α

First ($\alpha$) = First(b) = {b}

$\therefore M[\epsilon, b] : \epsilon \rightarrow b$

3b) Construct SLR parsing table for the grammar

$S' \rightarrow S$

$S \rightarrow AA$

$A \rightarrow aA \mid b$

Ans

Let $I = \{S' \rightarrow S\}$

then $I_0$ = closure(I)

$= \{S' \rightarrow .S$

$S \rightarrow .AA$

$A \rightarrow .aA$

$A \rightarrow .b\}$

$I_1$ = goto($I_0$, S)

$= \{S' \rightarrow S.\}$

$I_2$ = goto($I_0$, A)

$= \{S \rightarrow A$

$A \rightarrow .aA$

$A \rightarrow .b\}$

$I_3$ = goto($I_0$, a)

$= \{A \rightarrow a.A$

$A \rightarrow .aA$

$A \rightarrow .b\}$

$I_4$ = goto($I_0$, b)

$= \{A \rightarrow b.\}$

$I_5$ = goto($I_2$, A)

$= \{S \rightarrow AA.\}$

$I_6 = goto(I_2, a)$

$= \{A \to a \cdot A$

$A \to \cdot aA$

$A \to \cdot b\} = I_3$

$I_6 = goto(I_2, b)$

$= \{A \to b \cdot A\} = I_4$

$I_6 = goto[I_3, A]$

$= \{A \to aA \cdot\}$

$I_7 = goto(I_3, a)$

$= \{A \to a \cdot A$

$A \to \cdot aA$

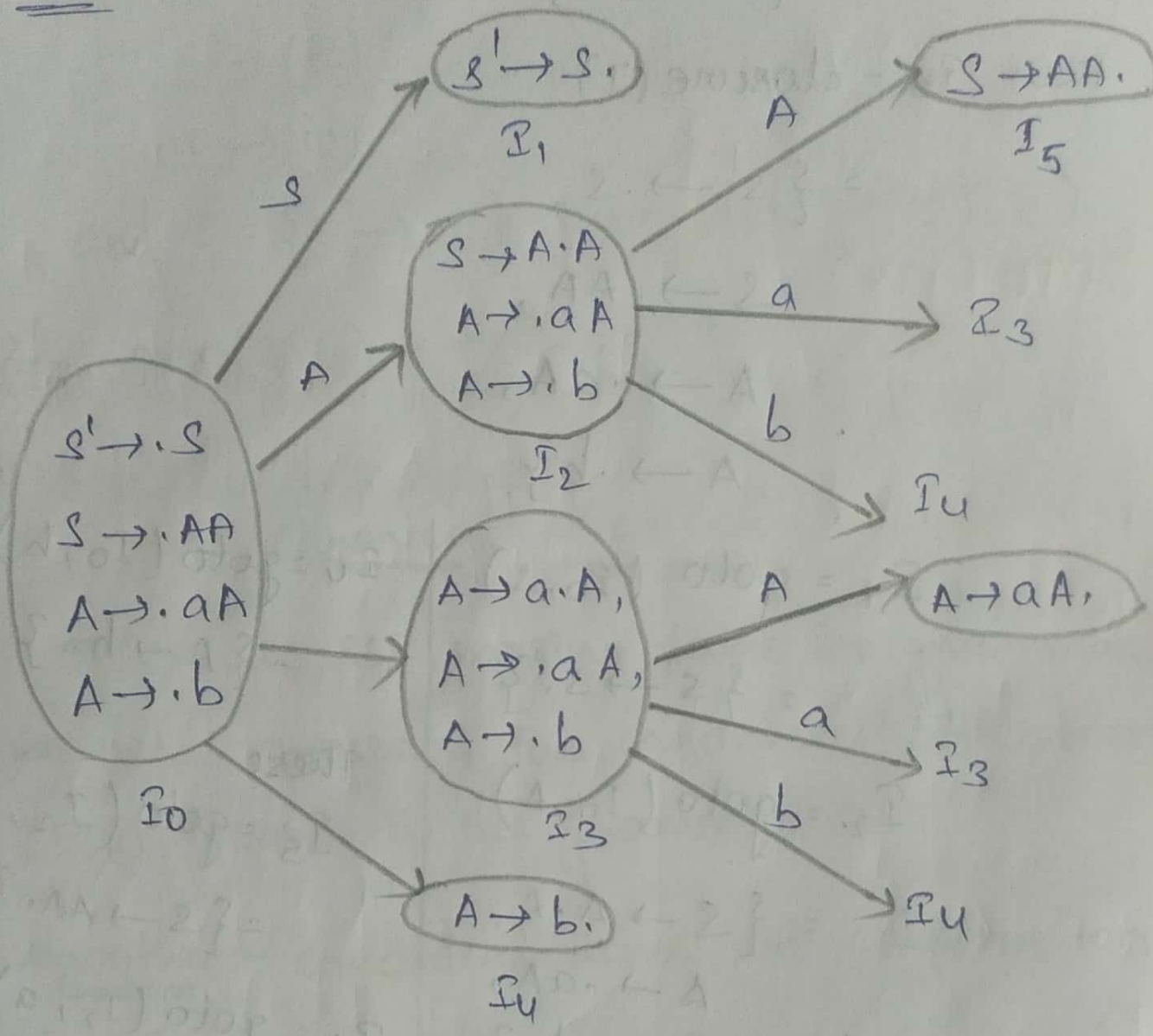
$A \to \cdot b\} = I_3$

$I_7 = goto(I_3, b)$

$= \{A \to b \cdot\} = I_4$

Here also $I_7$ is not generated

DFA



3a) construction SLR Parsing table:-

Algorithm:-

1. construct $C = \{I_0, I_1, \ldots I_n\}$, the collection of LR(0) items for G.
2. state i is constructed from state $I_i$.
   a) If $[A \to \alpha . a\beta]$ is in state $I_i$ and Goto$(i,a) = I_j$, then set ACTION $[i,a]$ to "shift j". Here 'a' is a terminal.
   b) If $[A \to \alpha .]$ is in state $I_i$, then set ACTION $[i,a]$ to "reduce $A \to \alpha$" for all a in follow (A).
   c) If $[S' \to S.]$ is in state $I_i$, then set ACTION $[i,\$]$ to "Accept".
3. If any conflicts appears then we say that the grammar is not suitable for SLR.
4. If GOTO $(I_i, A) = I_j$ then GOTO $[i,A] = j$
5. The initial state of the parser is the one constructed from the set of items containing $[S' \to .S]$.
6. All entries not defined by above rules are made "error".