Lecture 4

4-11-2024

* For the same string, if there is different syntax tree then the grammer is ambiguous.

$E \rightarrow E+T \mid E-T \mid T$

$T \rightarrow T*F \mid T/F \mid F$

$F \rightarrow id \mid number \mid (E)$

} Thus it removes the ambiguity of order of operations. mul must be done before addition

(ensuring addition before multiplication

* we need ~~PDA~~ to verify whether a string belongs to CFG.

* $E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow id \mid (E)$

input string : ~~id~~ + id * id $

(i) + id * id $

(ii) + id * id $

(iii) + id * id $

(iv) + id * id $

(v) id * id $

(vi) * id $

(vii) " , (viii) "

(ix) id $

(x) $

(xi) $ (xii) $ (xiii) $

stack

$ ← nothing in the stack

(1) $ id

(2) $ F

↑ reverse order derivation

③ $ T we are not pushing + sign since according to grammer we can only use + sign in E.

④ $ E

⑤ $ E +

⑥ $ E + id

⑦ $ E + F

⑧ $ E + T → we can convert it to E but we will not since * will not work then.

⑨ $ E + T *

⑩ $ E + T * id

⑪ $ E + T * F ← Bottom-up approach

⑫ $ E + T

⑬ $ E

↳ indicating the End

we ended up with empty input string & stack has only the start symbol of the grammer.

E ← 13 th step of stack

E + T ← 12 th

E + T * F ← 11 th

E + T * id

E + F * id

E + id * id

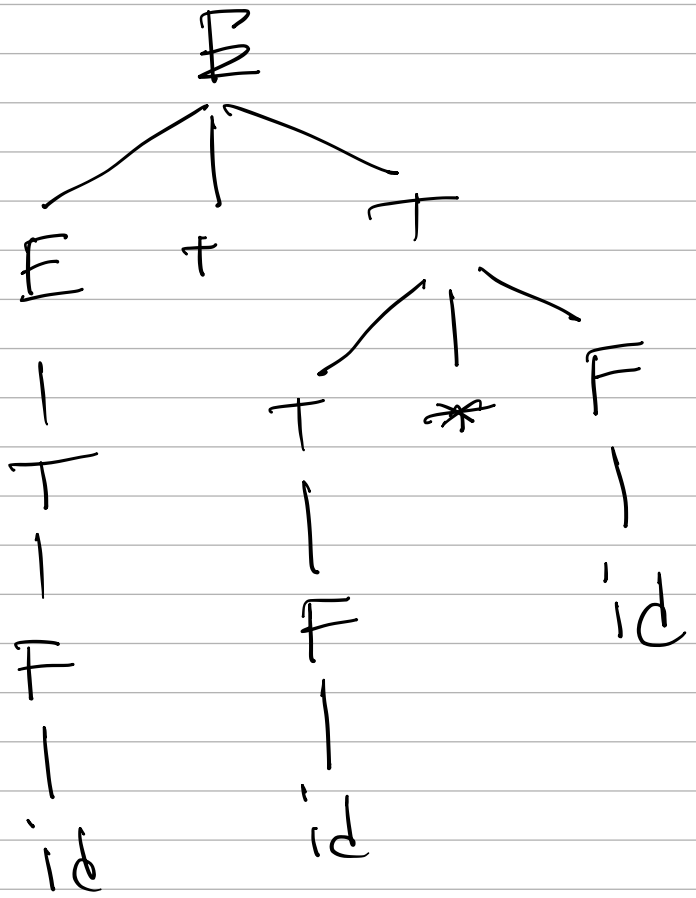
T + id * id

E + id * id

id + id * id

Right most derivation →

Syntax tree



* Syntax analyzer we create are called Bottom-up parsers / syntax analyzer.

* the input is reader from leftmost and the derivation is done from Rightmost;

∴ LR parser.

left-right scanning

right most derivation

↳ it generates the rightmost derivation in reverse order.

* Bottom-up parsers are also called LR parser.

* Top down is a small subset of Bottom up. That's why we use Bottom up.

* In the stacke we either:
  1. push some token inside ⇐ Shift operation
  2. Converted a production body into the head of the production. ⇐ reduce

$$E \longrightarrow E + T$$

Head Body

Head to Body → derivation/expansion
Body to Head → reverse derivation/reduce

* Bottom-up parsen is also called Shift-reduce parsen.
  * making the right choice between shift & reduce is a critical decision. we need an algorithm to do the task. at 8th step we shifted rather than reducing. To take the decision we looked into the remaining input string. we restrict the numbers of " " " to investigate since otherwise it will take too much time. So in the LR parsen we give the number of future tokens the syntax analyzen checks to make its decision, inside a paranthesis.

LR(1) parsen
↳ checks 1 future token

LR(2) parser

↳ checks 2

The complexity increases exponentially.

* We will learn simple LR(1) parser

↓

Simple LR parser

↓

CLR parser

* In lab we will use LALR parser.

↓

look ahead LR parser.

Steps to construct LR syntax analyzer:

① You augment the grammer with a new start symbol.

② You create an LR(0) automaton diagram from the augmented grammer.

③ Compute two functions (First and Follows) of non terminals.

④ Construct a two parts parsing table from information from step ② & ③.