

COMP 442 / 6421

Compiler Design

Tutorial 3

Instructor:

Dr. Joey Paquet

paquet@cse.concordia.ca

TAs:

Haotao Lai

h_lai@encs.concordia.ca

Jashanjot Singh

s_jashan@cs.concordia.ca



Recall Assignment2

1. Convert the given CFG to LL(1) grammar
 - a. Need to use tools to verify your converting procedure
 - b. Remove the grammar from EBNF to non-EBNF presentation
 - c. Remove ambiguity and left recursion
2. Implement a LL(1) parser
 - a. Recursive descent predictive parsing
 - b. Table-driven predictive parsing

In the following slides, I use step 1 to refer point No. 1, step 2 to refer point No. 2;

Question: What is exactly a LL parser and what is the 1 stand for?



Three Roadblocks in Assignment 2 Step 1

Quick review

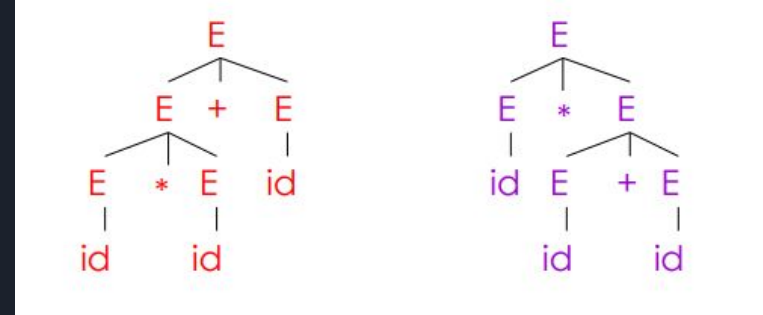
1. Ambiguity
2. Non-deterministic
3. Left recursion

For theoretical detail, see the lecture slide set [syntax analysis: introduction].

Ambiguity Grammar

Grammar: $E \rightarrow E + E \mid E * E \mid id$

Input string: id * id + id



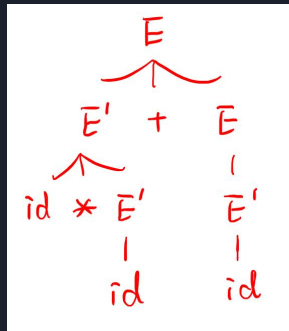
Requirement of the **parse tree**:

A tree that its in order traversal should give the string same as the input string

Ambiguity Grammar

The solution for ambiguity is rewrite the grammar (that's exactly what you need to do in assignment 2) to make it unambiguous.

In this case, we want to enforce precedence of multiplication over addition.



original: $E \rightarrow E + E \mid E * E \mid id$

modified:

$E \rightarrow E' + E \mid E'$

$E' \rightarrow id * E' \mid id$



Non-deterministic Grammar

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3$$

1. backtracking can solve this problem, but it is inefficient;
2. introduce a new non-terminal which we refer as left factoring

$$\begin{aligned} A &\rightarrow \alpha A' \\ A' &\rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \end{aligned}$$

Left Recursion

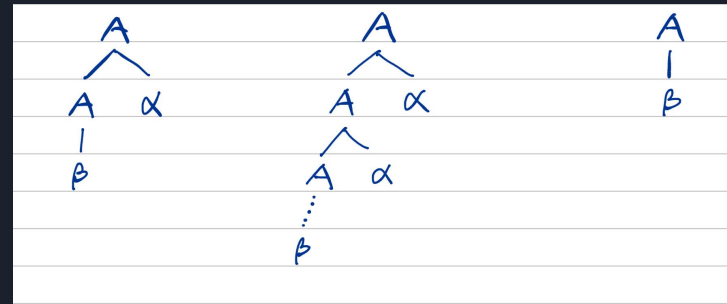
Grammar: $A \rightarrow A\alpha \mid \beta$

By analyzing these three possibilities, our goal is to construct something like: $A \rightarrow \beta\alpha^*$

But we don't allow $*$ in the grammar, so we can replace α^* with a new non-terminal A' , so we have:

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \epsilon$





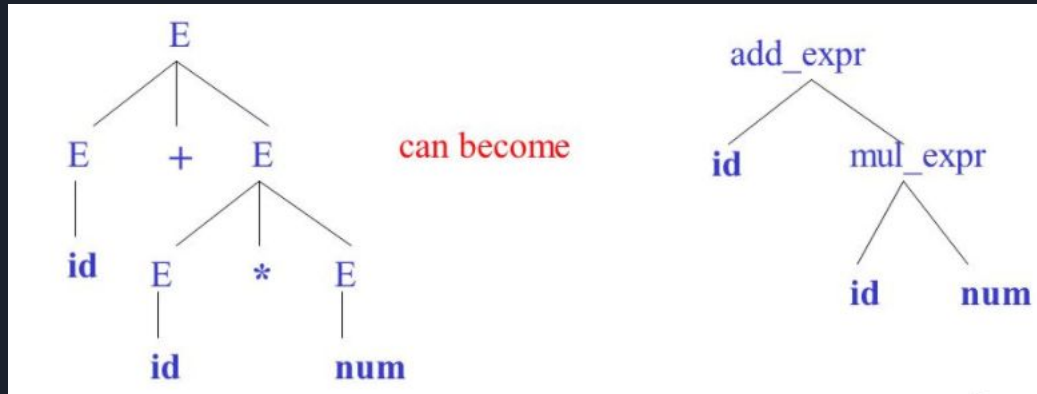
Big Tip

When you implement the assignment 2, the online tool we introduced in the tutorial 2 can solve the non-deterministic and left recursion problems!

But as the theoretical part of the course you should also be able to fix the grammar by hand.

Parse Tree

- concrete syntax tree (left)
- abstract syntax tree (right)





First Set and Follow Set

example 1:

$S \rightarrow ABCDE$

$A \rightarrow a \mid \epsilon$

$B \rightarrow b \mid \epsilon$

$C \rightarrow c$

$D \rightarrow d \mid \epsilon$

$E \rightarrow e \mid \epsilon$



First Set and Follow Set

example 2:

$S \rightarrow Bb \mid Cd$

$B \rightarrow aB \mid \epsilon$

$C \rightarrow cC \mid \epsilon$



First Set and Follow Set

example 3:

$S \rightarrow ACB \mid CbB \mid Ba$

$A \rightarrow da \mid BC$

$B \rightarrow g \mid \epsilon$

$C \rightarrow h \mid \epsilon$

Thanks