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

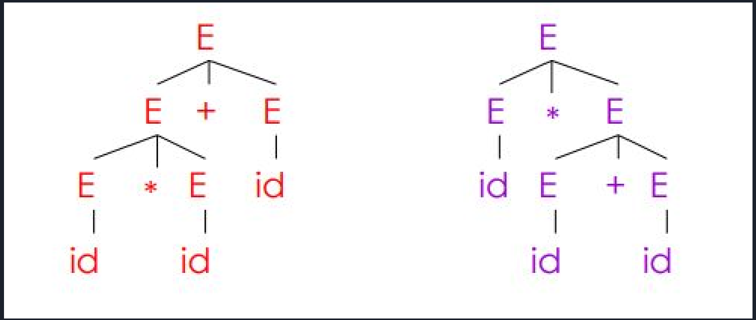
1. Ambiguity
2. Non-deterministic
3. Left recursion

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

Ambiguity Grammar

Grammar: E -> E + E | E * E | id

Input string: id * id + id



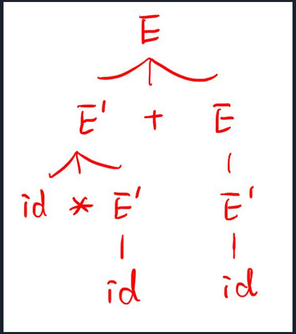
Requirement of the parse tree:

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 -> E + E | E * E | id

modified:

E -> E' + E | E'

E' -> id * E' | 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

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

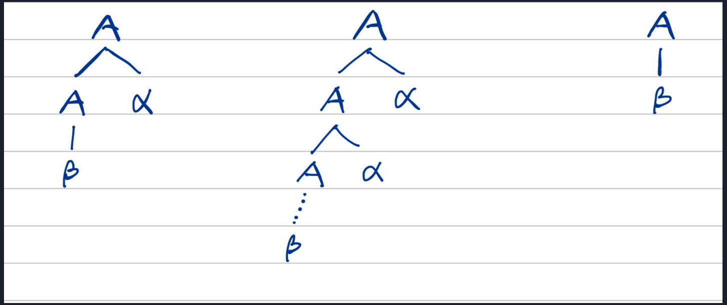
Left Recursion

Garmmar: A -> Aα | β

By analyze these three possibilities, our goal is to construct something like: A -> βα*

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

A -> βA'
A' -> αA' | ε



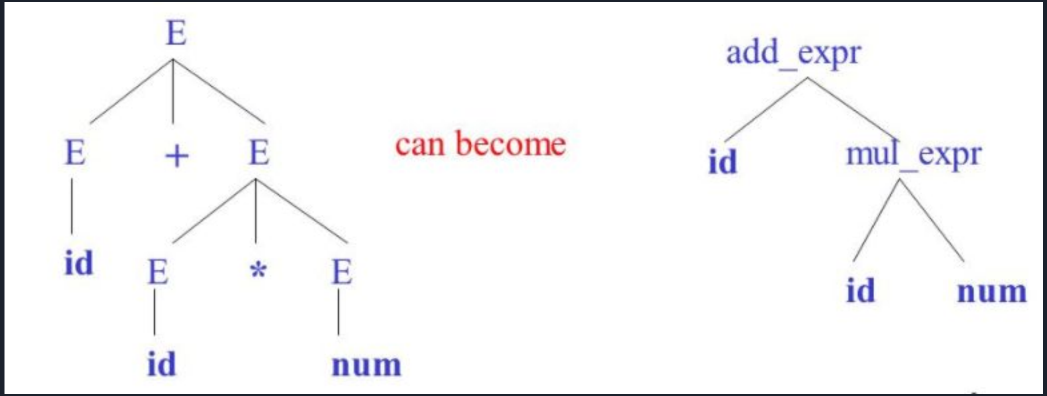
Big Tip

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 -> A B C D E
A -> a | ε
B -> b | ε
C -> c
D -> d | ε
E -> e | ε

First Set and Follow Set

example 2:

S -> B b | C d
B -> a B | ε
C -> c C | ε

First Set and Follow Set

example 3:

S -> A C B | C b B | B a
A -> d a | B C
B -> g | ε
C -> h | ε

Tool

If you plan to use the table-driven approach, you will need a parse table. Of course you can generate your own parse table, or put a proper grammar into a tool and it will give you the table.

We propose an online tool to do that:
http://hackingoff.com/compilers/ll-1-parser-generator

note: you need to use EPSILON to repersent ε

Thanks