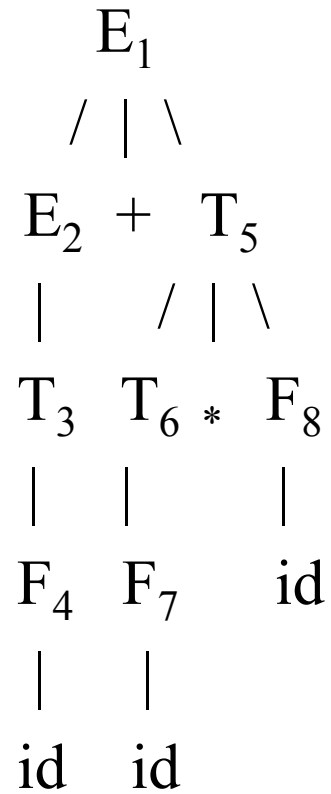


LL(1) Parsing

- LL(1) is a Top Down parsing scheme.
- Applies productions from goal symbol to derive grammar sentence.
- First L – Scanner moves from left to right.
- Second L – Parser expands leftmost non-terminal first.
- (1) – Scanner supplies parser with one terminal/token to figure out which production to apply next.

Top-Down Parse



Def: Unambiguous Grammar

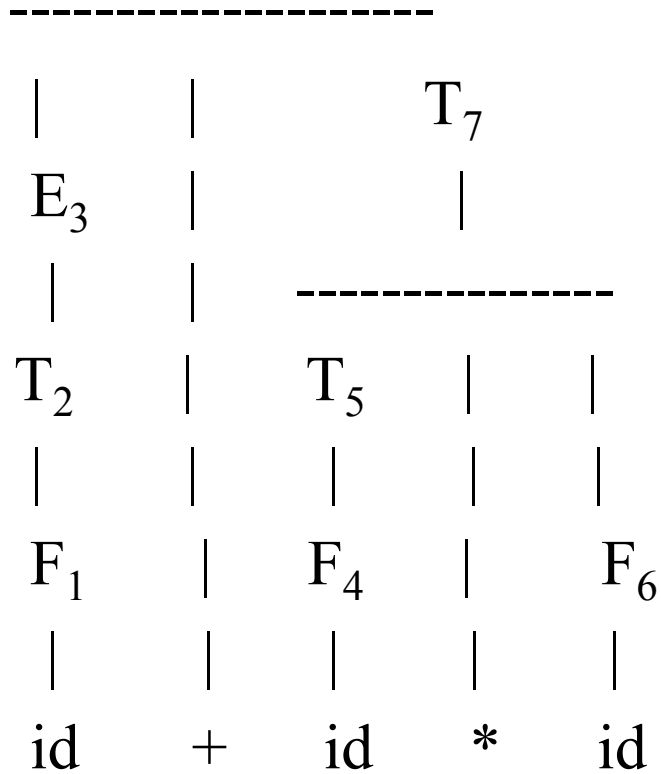
- Context Free.
- Every sentence in grammar has at most one leftmost derivation.

LR(1) Grammar

- LR(1) is a bottom-up parsing scheme.
- Reduces from grammar sentence to goal symbol using reductions.
- L – Scanner moves from left to right.
- R – Rightmost derivation in reverse.
- (1) – Scanner supplies parser with one terminal/token to figure out which reduction to apply next.

Bottom-Up Parse

E_8



Rightmost Derivation in Reverse

$$\begin{aligned} \text{id} + \text{id} * \text{id} &\rightarrow F_1 + \text{id} * \text{id} \rightarrow T_2 + \text{id} * \text{id} \rightarrow \\ E_3 + \text{id} * \text{id} &\rightarrow E + F_4 * \text{id} \rightarrow E + T_5 * \text{id} \rightarrow \\ E + T * F_6 &\rightarrow E + T_7 \rightarrow E_8 \end{aligned}$$

Notice that proceeding backwards, the rightmost non-terminal symbol (highest number) is expanded.

Grammar Design

Design an unambiguous context-free grammar G_1 where

- $+$, $-$, $*$, and $/$ are left associative
- $^$ (exponentiation) is right associative
- Precedence from lowest to highest is:
 - 1) Binary $+$, $-$
 - 2) Binary $*$, $/$
 - 3) Binary $^$
 - 4) Unary $-$

Grammar G_1

$$E \rightarrow E + T \mid E - T \mid T$$
$$T \rightarrow T * F \mid T / F \mid F$$
$$F \rightarrow P \wedge F \mid P$$
$$P \rightarrow -P \mid Q \quad \{\text{or } P \rightarrow -Q \mid Q\}$$
$$Q \rightarrow (E) \mid \text{id}$$

Semantic Actions

Grammar G_5

$$E \rightarrow E + T [a] \mid T$$
$$T \rightarrow T * F [m] \mid F$$
$$F \rightarrow (E) \mid \text{id} [p]$$

[a] = add – add top two elements of semantic stack and emit add instruction.

[m] = multiply – multiply top two elements of semantic stack and emit multiply instruction.

[p] = load current token in register R_i and push current token on semantic stack

Input String for G_2

$\$ id_0 + id_1 * id_2 \$$

Semantic Stack for G_2

		id_2		
	id_1	id_1	R_2	
id_0	id_0	id_0	id_0	R_2

Instructions Emitted by G_2

load R_0 id_0

load R_1 id_1

load R_2 id_2

mul R_1 R_2

add R_0 R_2