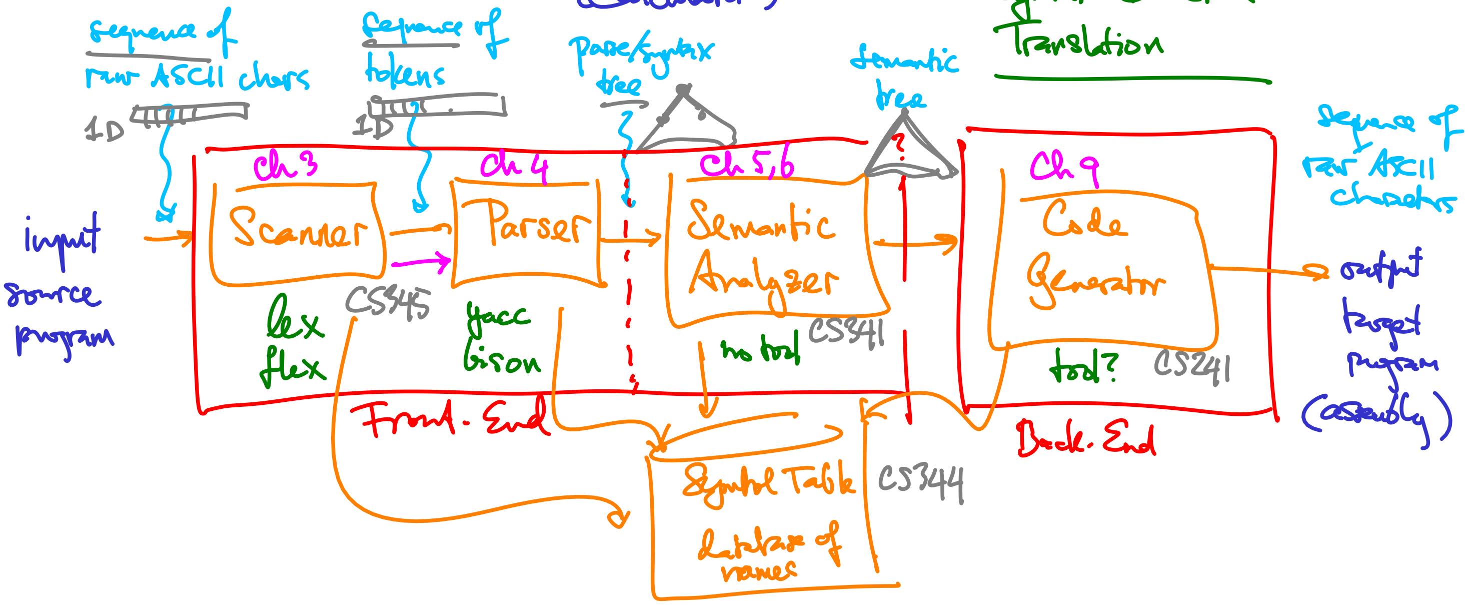
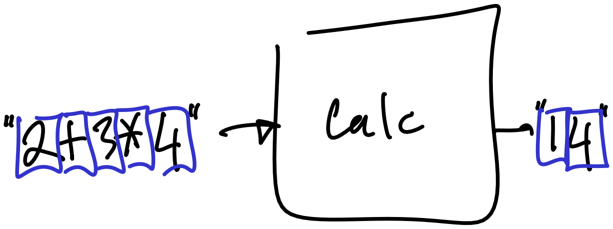
01/24/2019 Chapter 2 Simple One-Pass Compiler (Calculator) ⇒ Chapter 5 Syntax-Directed Translation



Calculator Grammar : language of simple arithmetic expressions

+ *



1. E → E + E
2. E → E * E
3. E → (E)
4. E → NUM

G1 $E \rightarrow E \underline{+} E \mid E \underline{*} E \mid \underline{(} E \underline{)} \mid \underline{NUM}$

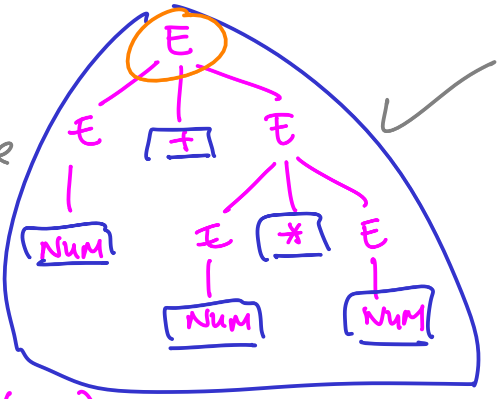
This grammar is ambiguous!

$NUM = [0-9]^{+} = (0|1|2|\ldots|9)(0|1|2|\ldots|9)^{*}$

Fix: precedence & associativity rules

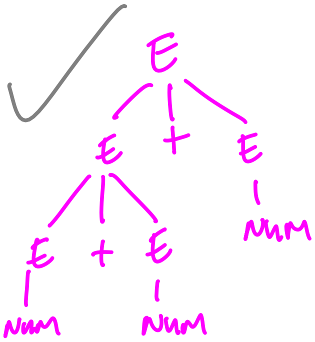
$$E \overset{1}{\Rightarrow} E + \underline{E}$$
$$\overset{3}{\Rightarrow} \underline{E} + E * E$$
$$\overset{4}{\Rightarrow} NUM + \underline{E} * E$$
$$\overset{4}{\Rightarrow} NUM + NUM * \underline{E}$$
$$\overset{4}{\Rightarrow} \underline{NUM + NUM * NUM}$$

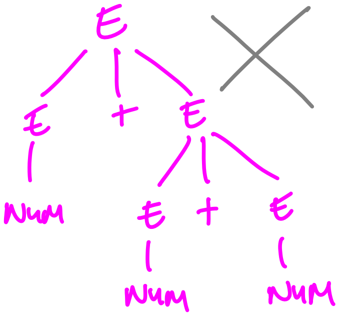
+ has lower precedence than *

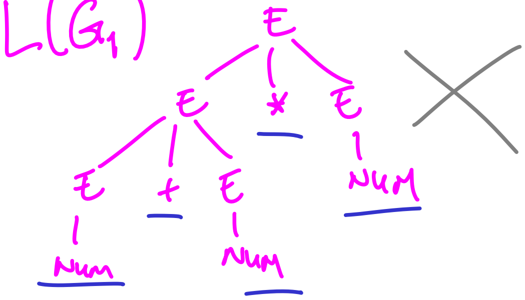


+ is left-associative

NUM + NUM * NUM ∈ L(G_1)







NUM + NUM + NUM

(2 − 3) − 4

2 − (3 − 4)

G2: R =

	associativity	precedence
E → E + T | T	left	low
T → T * F | F	left	med
F → (E) | NUM		high

↑ T or F

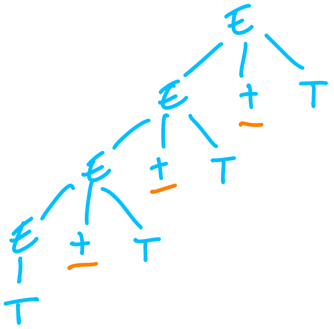
variables = {E, T, F}

terminals = {+, *, (,), NUM}

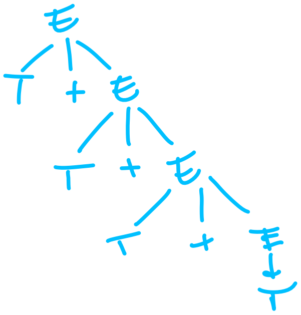
claim:

G_2 is not ambiguous

E → E + T | T



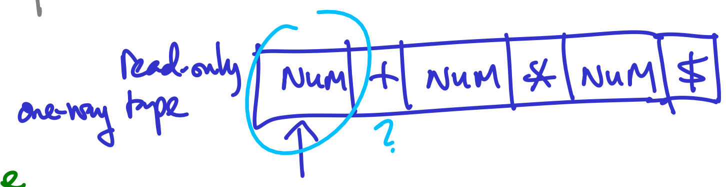
E → T + E | T

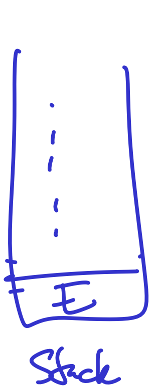


Idea: Recursive-Descent Parsing

⇓ top-down method

G_2: $E \to E+T \mid T$
$T \to T*F \mid F$
$F \to (E) \mid NUM$





method to apply:

- turn each variable into a function
- use the grammar rules as the body for the functions

```
E( )
{
    either
        [E( )
         See(+)
         T( )]
    or
        [T( )]
}
```

dangerous recursive call

```
T( )
{
    either
        [T( )
         see(*)
         F( )]
    or
        [F( )]
}
```

```
F( )
{
    if token == (
        [see(( )
         E( )
         see( ))]
    else if token == NUM
        [see(NUM)]
    else ERROR( )
}
```

Left Recursion Problem

$$A \to A\alpha \mid \beta$$

α is a sequence of grammar symbols (variables and/or terminals)

β that does not begin with A

ex: $\boxed{E \to E+T \mid T}$

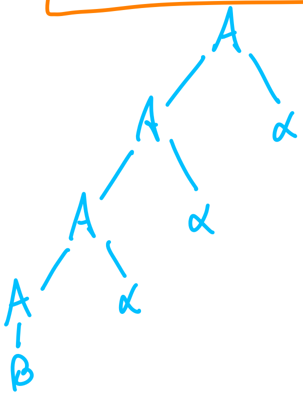
$A = E$ $\alpha = +T$ $\beta = T$

$\boxed{A \to A\alpha \mid \beta} \Longrightarrow$

fix

$$\begin{aligned} A &\to \beta A' \\ A' &\to \alpha A' \mid \varepsilon \end{aligned}$$

aux variable



$\beta\alpha^*$

how to fix $E \to E + \underbrace{+T}_{\alpha} \mid \underbrace{T}_{\beta} \Longrightarrow$

$$\begin{aligned} E &\to TE' \\ E' &\to \underline{+TE'} \mid \underline{\varepsilon} \end{aligned}$$

tail recursive loop

todo

```
E( )
{   T( );
    E'( );
}
```

```
E'( )
{   if (token == +)
        [see(+)
         T( )
         E'( )]
}   else return;
```