CS143: Parsing I

David L. Dill

Stanford University

Parsing (and lexical analysis wrapup)

- Practical Issues in Lexical Analysis (finish)
- Parsing
  - Context-Free Grammars
  - Parse Trees and Languages
  - Ambiguity
  - Mathematical Definitions
  - Precedence and Associativity
  - Extended BNF

Start Conditions

A better way to handle comments

When you see "/*"

Use new DFA

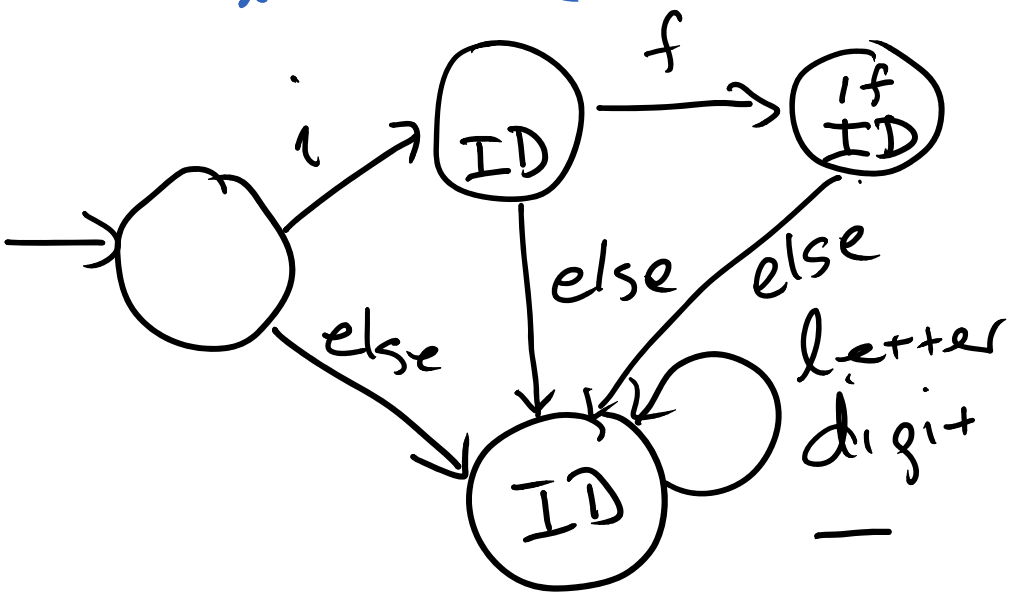
Patterns:

- any single char — discard
- "*/" - discard, return to main DFA

Rule Ordering

`if` `if`

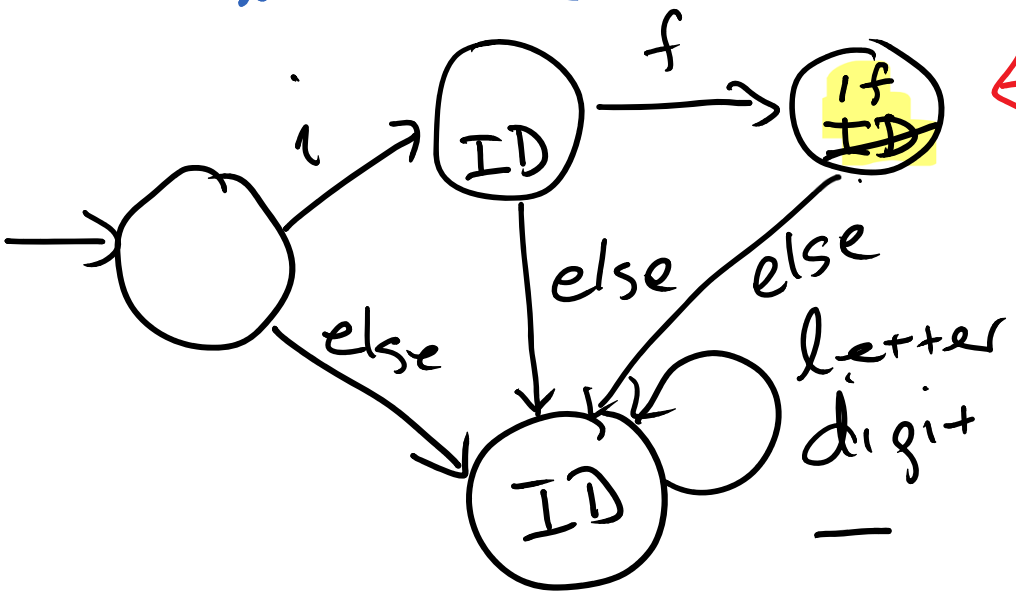
ID letter(letter | digit | _)



Rule Ordering

`if` `if` ← special case first

ID letter(letter | digit | _) ← default



← label for first rule in file wins.

Parsing

Context-Free Grammars

CFG for arithmetic expressions

$$E \rightarrow E + E$$
$$E \rightarrow E - E$$
$$E \rightarrow E \times E$$
$$E \rightarrow -E$$
$$E \rightarrow (E)$$
$$E \rightarrow \text{num}$$

↑ token type (any number)

CFG for arithmetic expressions

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E \times E$

$E \rightarrow -E$

$E \rightarrow (E)$

$E \rightarrow num$

production

LHS → RHS

(left-hand side) (right-hand side)

CFG for arithmetic expressions

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E \times E$

$E \rightarrow -E$

$E \rightarrow (E)$

$E \rightarrow \text{num}$

LHS of production must be a single symbol.

These symbols are called "nonterminals"

This grammar only has one, but a grammar can have many.

CFG for arithmetic expressions

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E \times E$

$E \rightarrow -E$

$E \rightarrow (E)$

$E \rightarrow \text{num}$

The sentence symbol is one of the non-terminals.

It is sort of like the start state of a DFA

In this case, it is E.

CFG for arithmetic expressions

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E \times E$

$E \rightarrow -E$

$E \rightarrow (E)$

$E \rightarrow num$

RHS is a string with 0 or more symbols.

Symbols that are not non terminals are terminals.

Here: +, −, ×, (,), num

Parse Trees and Languages

CFG for arithmetic expressions

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E \times E$

$E \rightarrow -E$

$E \rightarrow (E)$

$E \rightarrow$ num

Parse tree

1 + 2 x 3

E ← sentence symbol

CFG for arithmetic expressions

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E \times E$

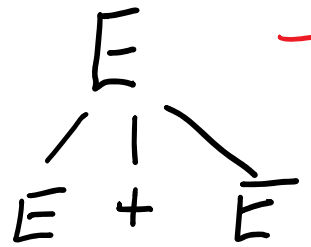
$E \rightarrow -E$

$E \rightarrow (E)$

$E \rightarrow num$

Parse tree

1 + 2 x 3



Parent = LHS

Children = RHS

CFG for arithmetic expressions

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E \times E$

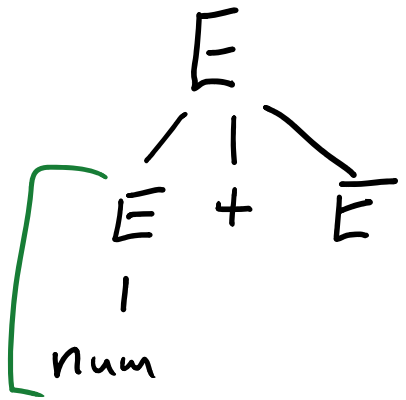
$E \rightarrow -E$

$E \rightarrow (E)$

$E \rightarrow num$

Parse tree

1 + 2 x 3



CFG for arithmetic expressions

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E \times E$

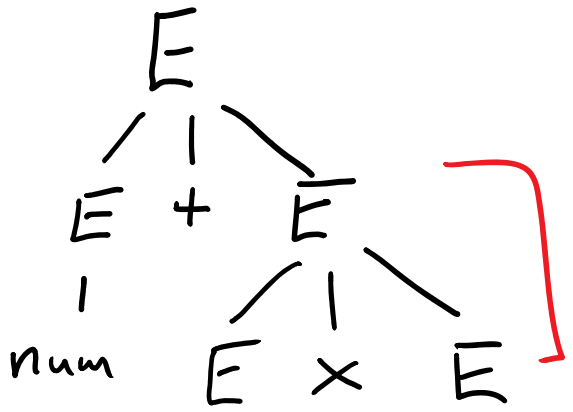
$E \rightarrow -E$

$E \rightarrow (E)$

$E \rightarrow num$

Parse tree

1 + 2 x 3



CFG for arithmetic expressions

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E \times E$

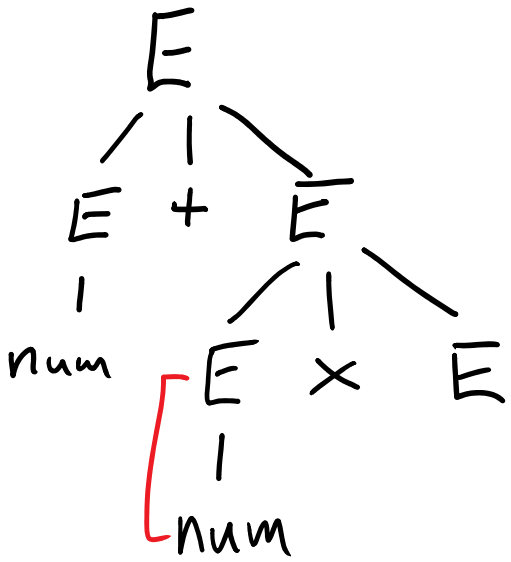
$E \rightarrow -E$

$E \rightarrow (E)$

$E \rightarrow num$

Parse tree

1 + 2 x 3



CFG for arithmetic expressions

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E \times E$

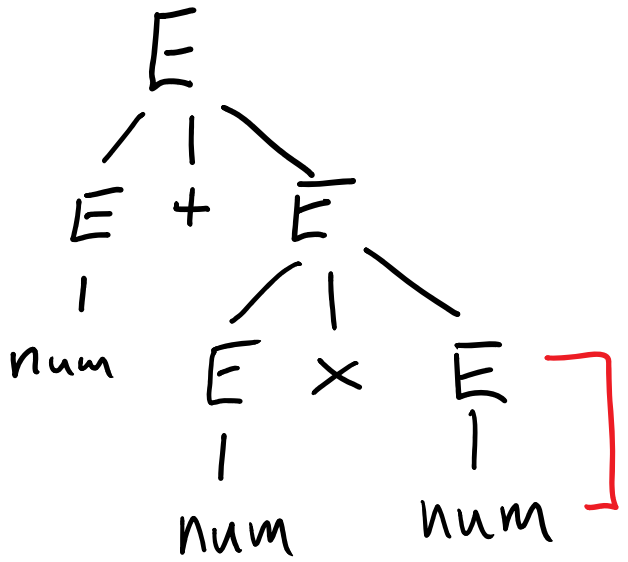
$E \rightarrow -E$

$E \rightarrow (E)$

$E \rightarrow num$

Parse tree

1 + 2 x 3



CFG for arithmetic expressions

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E \times E$

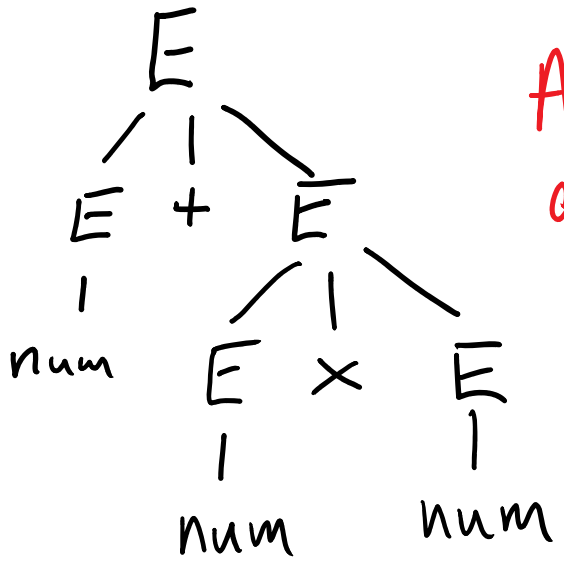
$E \rightarrow -E$

$E \rightarrow (E)$

$E \rightarrow num$

Parse tree

1 + 2 x 3



All leaves are terminals

CFG for arithmetic expressions

$E \to E + E$

$E \to E - E$

$E \to E \times E$

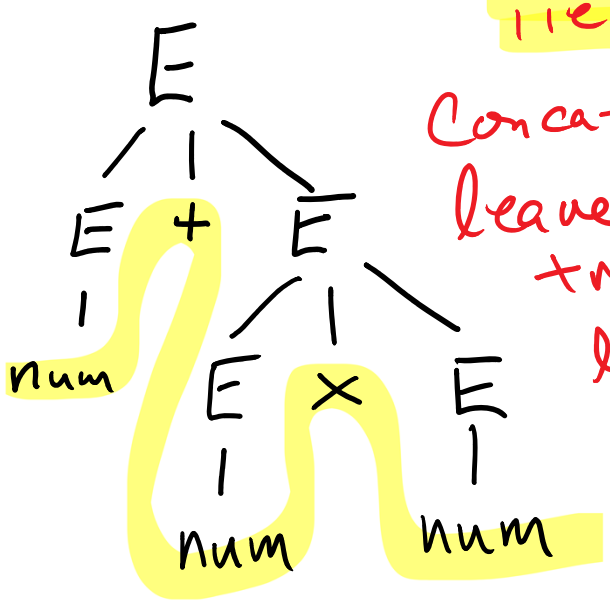
$E \to -E$

$E \to (E)$

$E \to num$

Parse tree

1 + 2 x 3



Yield

Concatenate leaves of tree from left to right.

num + num x num is in language of grammar.

Def The language of a CFG, G, is the set of all terminal strings that are the yield of some parse tree for the CFG.

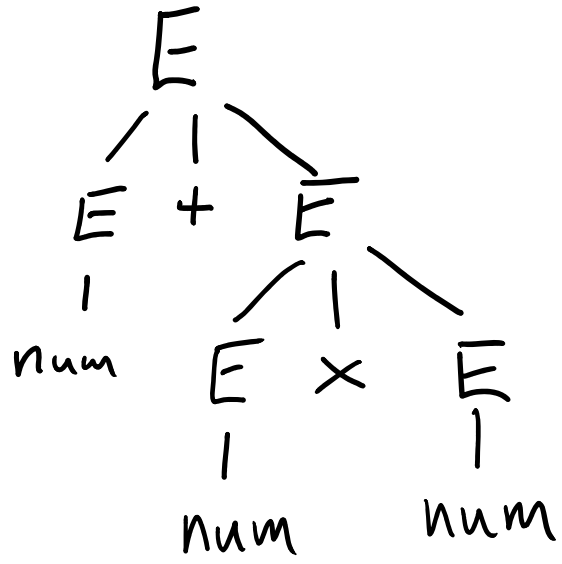
Parse tree for string x
= proof that x is in $L(G)$.

Def: The language of a CFG is a Context-Free Language (CFL).

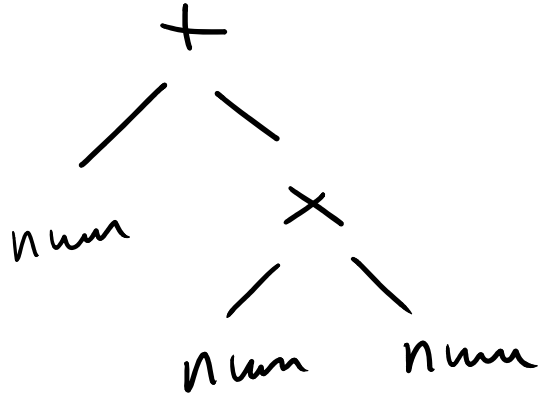
Parse Trees vs Abstract Syntax Trees

Parse tree

1 + 2 x 3



AST



"Abstract" = details suppressed

Ambiguity

Ambiguity

Def: A CFG is ambiguous if there is some terminal string with two parse trees.

num + num × num

Problems

1. Efficiency
   Multiple parses
2. Interpretation

$1 + \underline{2 \times 3} = 7$

$\underline{1 + 2} \times 3 = 9$

Ambiguity in CFGs is a hard problem.

- Above examples are easy - come from failure to specify precedence.
- There are many ways to be ambiguous
- Some CFLs are inherently ambiguous
- Ambiguity of a CFG is undecidable!

Mathematical Definitions

A CFG is (V, Σ, R, S)

V – finite set of nonterminal symbols ("variables")

Σ – finite set of terminal symbols

$R \subseteq V \times (V \cup \Sigma)^*$ – finite set of productions

$S \in V$ – sentence symbol.

Notation

$a, b, c, 0, 1$ — usually terminals

A, B, C, S — usually nonterminals

x, y, z, w — usually strings in Σ^*

α, β, σ — usually strings in $(\Sigma \cup V)^*$

Derivation step

$$\alpha A \beta \Rightarrow \alpha \gamma \beta \text{ if } A \to \gamma$$

Derivation: Sequence of 0 or more steps

$$E \Rightarrow E + E \Rightarrow E + E \times E \Rightarrow E + num \times E$$

$$\Rightarrow num + num \times E \Rightarrow num + num \times num$$

$\alpha \overset{*}{\Rightarrow} \beta$ (α "derives" β) if there is a derivation $\alpha \Rightarrow \alpha_0 \Rightarrow \dots \Rightarrow \beta$.

$\alpha \overset{*}{\Rightarrow} \alpha$ always

$\alpha \overset{*}{\Rightarrow} \beta$ (α "derives" β) if there is a derivation $\alpha \Rightarrow \alpha_0 \Rightarrow \cdots \Rightarrow \beta$.

$\alpha \overset{*}{\Rightarrow} \alpha$ always

Def The language of a CFG G ($L(G)$) is $\{x \mid x \in \Sigma^* \wedge S \overset{*}{\Rightarrow} x\}$

- x: "sentences"
- $x \in \Sigma^*$: terminal string
- $S \overset{*}{\Rightarrow} x$: derived from sentence symbol

Leftmost derivation

There are many derivations for one parse tree

$E \Rightarrow E + E \Rightarrow num + E \Rightarrow \ldots$

$E \Rightarrow E + E \Rightarrow E + E \times E \Rightarrow \ldots$

Def: A leftmost derivation is a derivation in which the leftmost nonterminal is replaced at each step.

Left most derivation

$E \Rightarrow E + E \Rightarrow num + E \Rightarrow num + E \times E$

$\Rightarrow num + num \times E \Rightarrow num + Num \times num$

A CFG is ambiguous iff there exists a string $x \in \Sigma^*$ that has

- 2 leftmost derivations,
  equivalently,
- 2 rightmost derivations
  equivalently
- 2 parse trees.

Two Leftmost Derivations for Same String

$E \Rightarrow E + E \Rightarrow num + E \Rightarrow num + E \times E$

$\Rightarrow num + num \times E \Rightarrow num + num \times num$

$E \Rightarrow E \times E \Rightarrow E + E \times E \Rightarrow num + E \times E$

$\Rightarrow num + num \times E \Rightarrow num + num \times num$

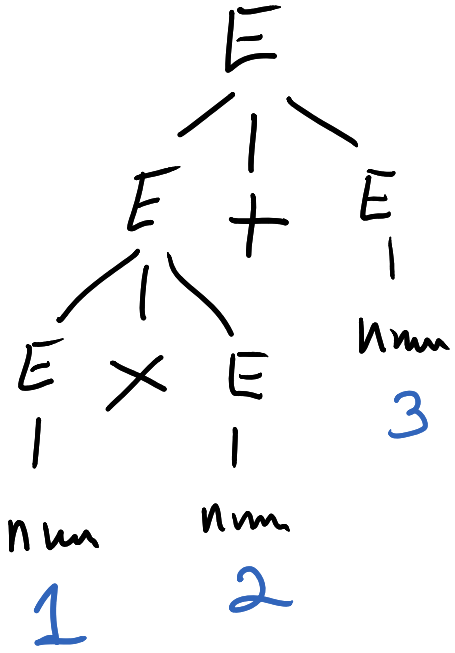
Announcements

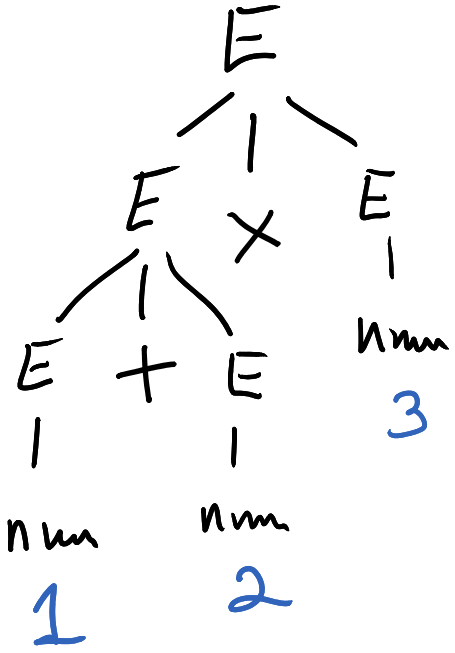
- Correction: PA1 is due 4/16.
- WA1 assigned today, due 4/16 (1 week).
  - “Automatic extension” without penalty until Monday 4/20
  - Other assign/due dates will remain the same.

Precedence and Association

$1 \times 2 + 3$

Which operator wins?





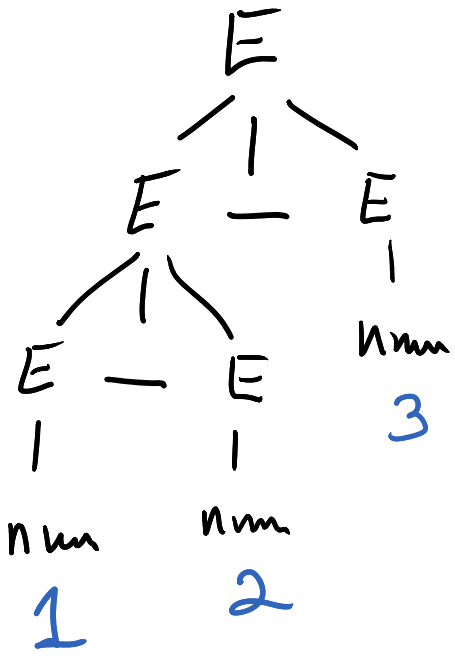
"higher precedence" means "stickier"

(1 × 2) + 3

↑ × "stickier"

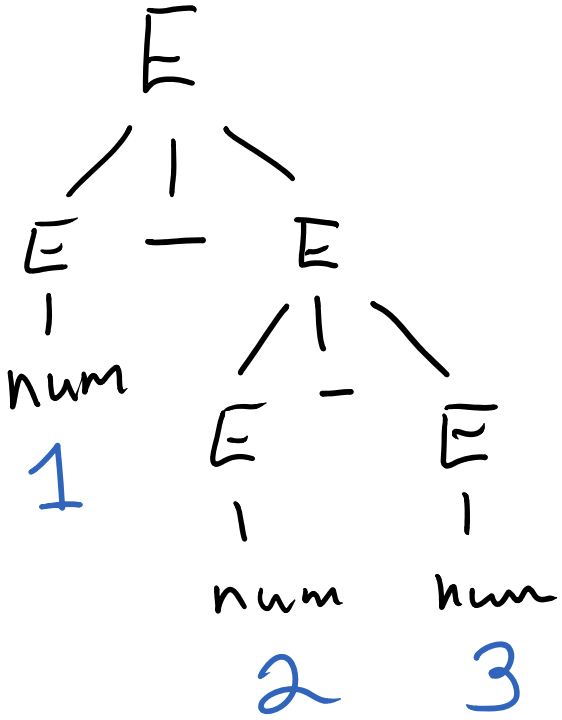
(1 + 2) × 3

↑ + stickier



(1 - 2) - 3

left associative



1 - (2 - 3)

right associative

CFG for arithmetic expressions

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E \times E$

$E \rightarrow -E$

$E \rightarrow (E)$

$E \rightarrow \text{num}$

This CFG has ambiguities for relative precedence of operators and left-right associativity

Abbreviated Notation.

$E \to E + E$

$E \to E - E$

$E \to E \times E$

$E \to -E$

$E \to (E)$

$E \to num$

Group productions with same LHS together

$E \to E+E \mid E-E \mid E \times E \mid -E \mid (E) \mid num$

Changing a CFG to fix precedence problems

Not always possible

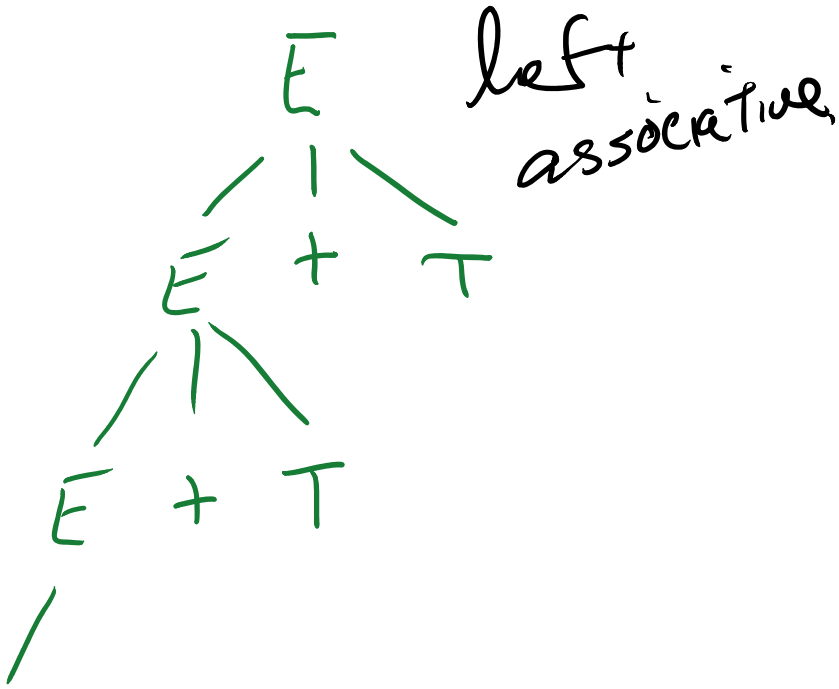
- Rank operators by increasing precedence
- New non-terminal for each precedence level
- Sentence symbol goes to lowest precedence operator
- Productions for generating more of same precedence operators, going to higher precedence
- left/right recursion for left/right associativity.

+, − same precedence (lowest)

× next higher precedence

unary − highest precedence.

$E \to E + T \mid E - T \mid T$



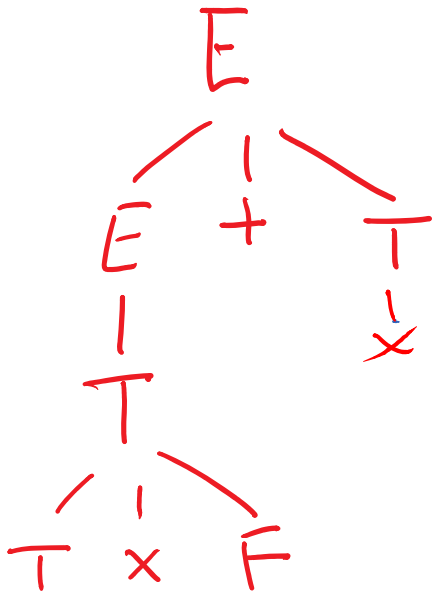
+, − same precedence (lowest)

× next higher precedence

unary − highest precedence.

$E \to E + T \mid E - T \mid T$

$T \to T \times F \mid F$



+, − same precedence (lowest)

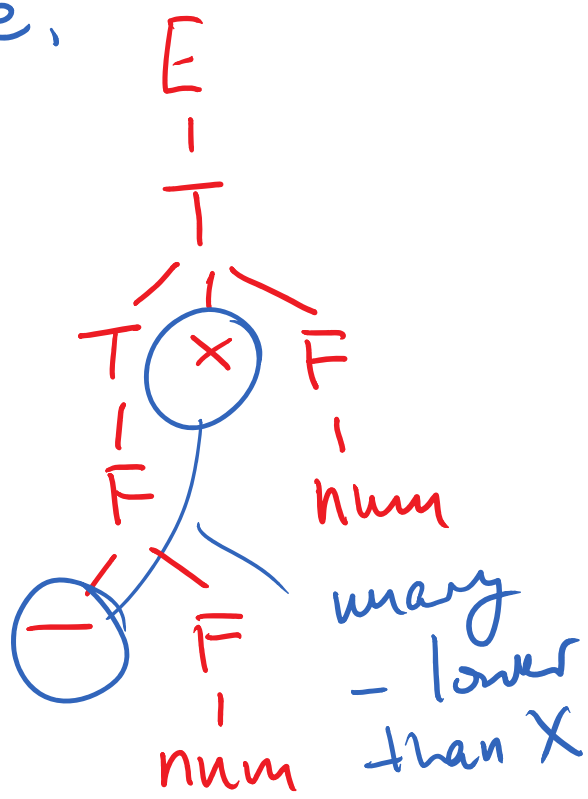
× next higher precedence

unary − highest precedence.

$E \to E + T \mid E - T \mid T$

$T \to T \times F \mid F$

$F \to -F \mid (E) \mid num$

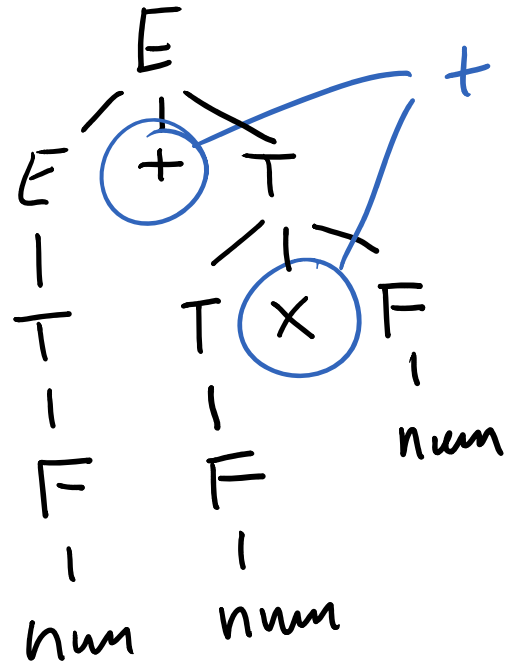


$E \to E + T \mid E - T \mid T$

$T \to T \times F \mid F$

$F \to -F \mid (E) \mid num$

num + num x num



+ always above x (unless parenthesized)

Extended BNF

EBNF

BNF - "Backus-Naur Form" = CFG

Idea: Allow regular expressions on RHS of productions.

Widely used (esp. in programming language definitions)

Parser generators usually require "plain CFG"

Need to convert EBNF → CFG

$A \to B^* d$ ← meta-symbol, not a terminal

0 or more B's on RHS

α^* abbreviates

$C \to C\alpha \mid \varepsilon$

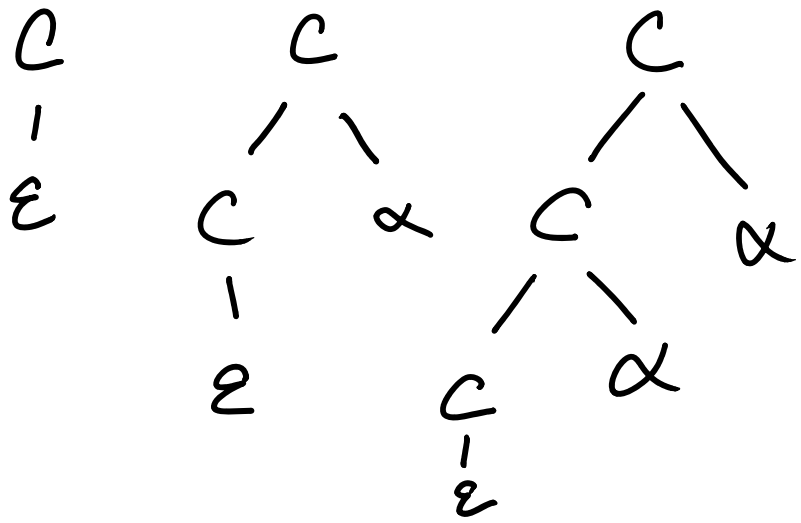
new non-terminal, not already in V.

$A \rightarrow B^* d$ ← meta-symbol, not a terminal

0 or more B's on RHS

α^* abbreviates

$C \rightarrow C\alpha \mid \varepsilon$



etc,

$A \rightarrow B^* d$

$A \rightarrow C d$

$C \rightarrow CB \mid \varepsilon$

α^*

$C \to C\,\alpha \mid \varepsilon$

↑ left-recursive

$C \to \alpha\,C \mid \varepsilon$

↑ right-recursive

May choose one or the other because

- Necessary for parsing method
- Grammar properties
- Tree shape.

α^+

$C \to C\alpha \mid \alpha$ $\quad$ $C \to \alpha C \mid \alpha$

α^+

$C \to C\alpha \mid \alpha \qquad C \to \alpha C \mid \alpha$

$\alpha?$ ($[\alpha]$ in COOL manual)

α^+

$C \to C\alpha \mid \alpha$ $\quad$ $C \to \alpha C \mid \alpha$

$\alpha ?$ ($[\alpha]$ in COOL manual)

$C \to \alpha \mid \varepsilon$