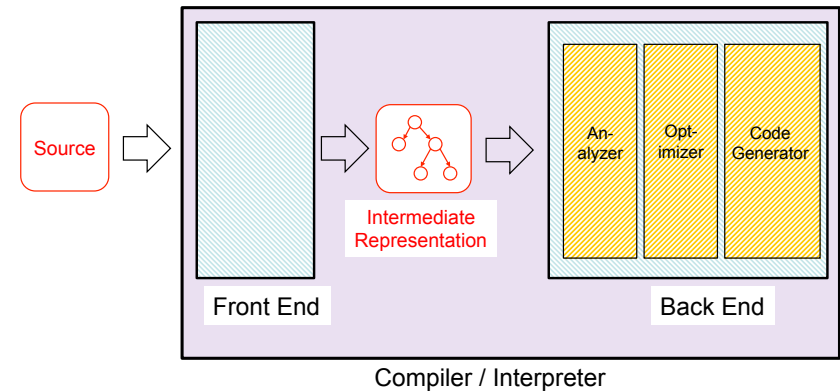


CMSC 330: Organization of Programming Languages

Context Free Grammars

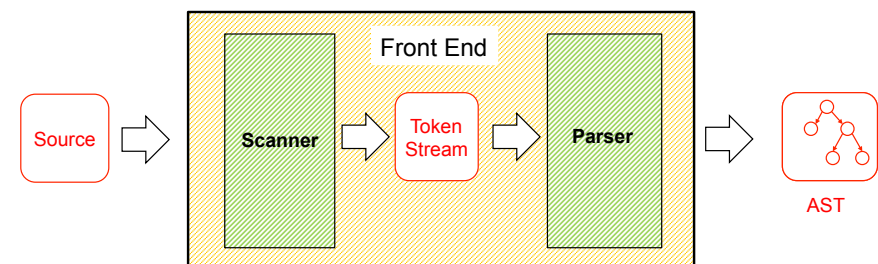
Architecture of Compilers, Interpreters



Implementing the Front End

- ▶ Goal: Convert program text into an AST
 - Abstract Syntax Tree
- ▶ ASTs are easier to work with
 - Analyze, optimize, execute the program
- ▶ Idea: Do this using regular expressions?
 - **Won't work!**
 - Regular expressions cannot reliably parse paired braces `{ { ... } }`, parentheses `((...))`, etc.
- ▶ Instead: Regexp for tokens (**scanning**), and **Context Free Grammars** for **parsing** tokens

Front End – Scanner and Parser



- **Scanner / lexer** converts program source into **tokens** (keywords, variable names, operators, numbers, etc.) using regular expressions
- **Parser** converts sequence of tokens into **AST** (abstract syntax tree) using context free grammars.

Context Free Grammar (CFG)

- ▶ A way of describing **sets of strings** (= languages)
 - The notation $L(G)$ denotes the language of strings defined by grammar G
- ▶ Example grammar: $S \rightarrow 0S \mid 1S \mid \epsilon$
String $s' \in L(G)$ iff
 - $s' = \epsilon$, or $\exists s \in L(G)$ such that $s' = 0s$, or $s' = 1s$
- ▶ Grammar is same as regular expression $(0|1)^*$
 - Generates / accepts the same set of strings

Parsing with CFGs

- ▶ CFGs formally define languages, but they do not define an *algorithm* for accepting strings
- ▶ Several styles of algorithm; each works only for less expressive forms of CFG
 - LL(k) parsing
 - LR(k) parsing
 - LALR(k) parsing
 - SLR(k) parsing
- ▶ Tools exist for building parsers from grammars
 - JavaCC, Yacc, etc.

Context-Free Grammars (CFGs)

- ▶ But CFGs can do what regexps cannot
 - $S \rightarrow (S) \mid \epsilon$ // represents balanced pairs of $()$'s
- ▶ In fact, CFGs subsume REs, DFAs, NFAs
 - There is a CFG that generates any regular language
 - But REs are a better notation for regular languages
- ▶ CFGs can specify programming language syntax
 - CFGs (mostly) describe the parsing process

Formal Definition: Context-Free Grammar

- ▶ A CFG G is a 4-tuple (Σ, N, P, S)
 - Σ – alphabet (finite set of symbols, or terminals)
 - ▶ Often written in lowercase
 - N – a finite, nonempty set of nonterminal symbols
 - ▶ Often written in uppercase
 - ▶ It must be that $N \cap \Sigma = \emptyset$
 - P – a set of productions of the form $N \rightarrow (\Sigma|N)^*$
 - ▶ Informally: the nonterminal can be replaced by the string of zero or more terminals / nonterminals to the right of the \rightarrow
 - ▶ Can think of productions as **rewriting rules** (more later)
 - $S \in N$ – the start symbol

Notational Shortcuts

- ▶ A production is of the form
 - left-hand side (LHS) \rightarrow right hand side (RHS)
- ▶ If not specified
 - Assume LHS of first production is the start symbol
- ▶ Productions with the same LHS
 - Are usually combined with |
- ▶ If a production has an empty RHS
 - It means the RHS is ϵ

```
S  $\rightarrow$  ABC // S is start symbol
A  $\rightarrow$  aA
  | b           // A  $\rightarrow$  b
  |           // A  $\rightarrow$   $\epsilon$ 
```

CMSC 330

9

Backus-Naur Form

- ▶ Context-free grammar production rules are also called Backus-Naur Form or **BNF**
 - A production like $A \rightarrow B c D$ is written in BNF as $\langle A \rangle ::= \langle B \rangle c \langle D \rangle$ (Non-terminals written with angle brackets and $::=$ instead of \rightarrow)
 - Often used to describe language syntax
- ▶ BNF was designed by
 - John Backus
 - ▶ Chair of the Algol committee in the early 1960s
 - Peter Naur
 - ▶ Secretary of the committee, who used this notation to describe Algol in 1962

CMSC 330

10

Generating Strings

- ▶ We can think of a grammar as **generating** strings by rewriting
- ▶ Example grammar:

$S \rightarrow 0S$

$S \rightarrow 1S$

$S \rightarrow \epsilon$

- $S \Rightarrow 0S$ // using $S \rightarrow 0S$
- $\Rightarrow 01S$ // using $S \rightarrow 1S$
- $\Rightarrow 011S$ // using $S \rightarrow 1S$
- $\Rightarrow 011$ // using $S \rightarrow \epsilon$

CMSC 330

11

Accepting Strings (Informally)

- ▶ Determining if $s \in L(S)$ is called **acceptance**: goal is to find a rewriting from S that yields s
 - $011 \in L(S)$ according to the previous rewriting
 - A rewriting is some sequence of productions (**rewrites**) applied starting at the start symbol
- ▶ Terminology
 - Such a sequence of rewrites is a **derivation** or **parse**
 - Discovering the derivation is called **parsing**

CMSC 330

12

Derivations

▶ Notation

- \Rightarrow indicates a derivation of one step
- \Rightarrow^+ indicates a derivation of one or more steps
- \Rightarrow^* indicates a derivation of zero or more steps

▶ Example

- $S \rightarrow 0S \quad S \rightarrow 1S \quad S \rightarrow \epsilon$

▶ For the string 010

- $S \Rightarrow 0S \Rightarrow 01S \Rightarrow 010S \Rightarrow 010$
- $S \Rightarrow^+ 010$
- $S \Rightarrow^* S$

Language Generated by Grammar

▶ $L(G)$ the language defined by G is

$$L(G) = \{ \omega \in \Sigma^* \mid S \Rightarrow^+ \omega \}$$

- S is the start symbol of the grammar
- Σ is the alphabet for that grammar

▶ In other words

- All strings over Σ that can be derived from the start symbol via one or more productions

Practice

▶ Try to make a grammar which accepts

- $0^*|1^*$ – 0^n1^n where $n \geq 0$ – 0^n1^m where $m \leq n$

$S \rightarrow A \mid B$
 $A \rightarrow 0A \mid \epsilon \quad S \rightarrow 0S1 \mid \epsilon \quad S \rightarrow 0S1 \mid 0S \mid \epsilon$
 $B \rightarrow 1B \mid \epsilon$

▶ Give some example strings from this language

- $S \rightarrow 0 \mid 1S$
> 0, 10, 110, 1110, 11110, ...
- What language is it, as a regexp?
> 1^*0

Example: Arithmetic Expressions

▶ $E \rightarrow a \mid b \mid c \mid E+E \mid E-E \mid E^*E \mid (E)$

- An expression E is either a letter a , b , or c
- Or an E followed by $+$ followed by an E
- etc...

▶ This **describes** (or **generates**) a set of strings

- $\{a, b, c, a+b, a+a, a^*c, a-(b^*a), c^*(b+a), \dots\}$

▶ Example strings not in the language

- $d, c(a), a+, b^{**}c$, etc.

Formal Description of Example

- Formally, the grammar we just showed is
 - $\Sigma = \{ +, -, *, (,), a, b, c \}$ // terminals
 - $N = \{ E \}$ // nonterminals
 - $P = \{ \begin{array}{l} E \rightarrow a, E \rightarrow b, E \rightarrow c, \\ E \rightarrow E-E, E \rightarrow E+E, \\ E \rightarrow E^*E, \\ E \rightarrow (E) \end{array} \}$ // productions
 - $S = E$ // start symbol

CMSC 330

17

(Non-)Uniqueness of Grammars

- Different grammars generate the same set of strings (language)
- The following grammar generates the same set of strings as the previous grammar
 - $E \rightarrow E+T \mid E-T \mid T$
 - $T \rightarrow T^*P \mid P$
 - $P \rightarrow (E) \mid a \mid b \mid c$

CMSC 330

18

Practice

- Given the grammar
 - $S \rightarrow aS \mid T$
 - $T \rightarrow bT \mid U$
 - $U \rightarrow cU \mid \epsilon$
- Provide derivations for the following strings
 - b $S \Rightarrow T \Rightarrow bT \Rightarrow bU \Rightarrow b$
 - ac $S \Rightarrow aS \Rightarrow aT \Rightarrow aU \Rightarrow acU \Rightarrow ac$
 - bbc $S \Rightarrow T \Rightarrow bT \Rightarrow bbT \Rightarrow bbU \Rightarrow bbcU \Rightarrow bbc$
- Does the grammar generate the following?
 - $S \Rightarrow^+ ccc$ Yes $S \Rightarrow^+ bS$ No
 - $S \Rightarrow^+ bab$ No $S \Rightarrow^+ Ta$ No

CMSC 330

19

Practice

- Given the grammar
 - $S \rightarrow aS \mid T$
 - $T \rightarrow bT \mid U$
 - $U \rightarrow cU \mid \epsilon$
- Name language accepted by grammar
 - $a^*b^*c^*$
- Give a different grammar accepting language
 - $S \rightarrow ABC$
 - $A \rightarrow aA \mid \epsilon$ // a^*
 - $B \rightarrow bB \mid \epsilon$ // b^*
 - $C \rightarrow cC \mid \epsilon$ // c^*

CMSC 330

20

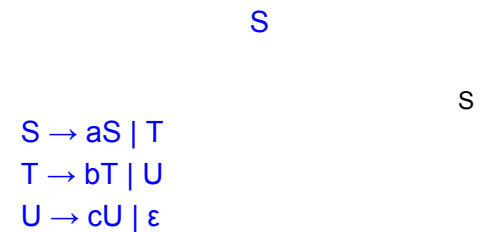
Parse Trees

- ▶ Parse tree shows how a string is produced by a grammar
 - Root node is the start symbol
 - Every internal node is a nonterminal
 - Children of an internal node
 - > Are symbols on RHS of production applied to nonterminal
 - Every leaf node is a terminal or ϵ
- ▶ Reading the leaves left to right
 - Shows the string corresponding to the tree

CMSC 330

21

Parse Tree Example



CMSC 330

22

Parse Tree Example

$S \Rightarrow aS$

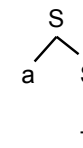
$$\begin{aligned} S &\rightarrow aS \mid T \\ T &\rightarrow bT \mid U \\ U &\rightarrow cU \mid \epsilon \end{aligned}$$


CMSC 330

23

Parse Tree Example

$S \Rightarrow aS \Rightarrow aT$

$$\begin{aligned} S &\rightarrow aS \mid T \\ T &\rightarrow bT \mid U \\ U &\rightarrow cU \mid \epsilon \end{aligned}$$


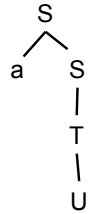
CMSC 330

24

Parse Tree Example

$S \Rightarrow aS \Rightarrow aT \Rightarrow aU$

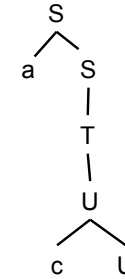
$S \rightarrow aS \mid T$
 $T \rightarrow bT \mid U$
 $U \rightarrow cU \mid \epsilon$



Parse Tree Example

$S \Rightarrow aS \Rightarrow aT \Rightarrow aU \Rightarrow acU$

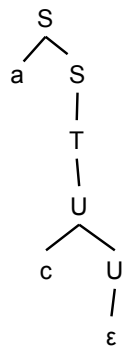
$S \rightarrow aS \mid T$
 $T \rightarrow bT \mid U$
 $U \rightarrow cU \mid \epsilon$



Parse Tree Example

$S \Rightarrow aS \Rightarrow aT \Rightarrow aU \Rightarrow acU \Rightarrow ac$

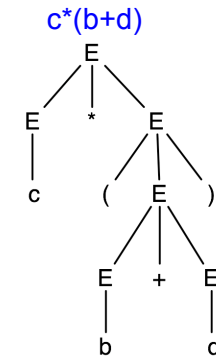
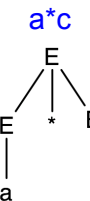
$S \rightarrow aS \mid T$
 $T \rightarrow bT \mid U$
 $U \rightarrow cU \mid \epsilon$



Parse Trees for Expressions

- A **parse tree** shows the structure of an expression as it corresponds to a grammar

$E \rightarrow a \mid b \mid c \mid d \mid E+E \mid E-E \mid E^*E \mid (E)$



Practice

$E \rightarrow a \mid b \mid c \mid d \mid E+E \mid E-E \mid E^*E \mid (E)$

Make a parse tree for...

- a^*b
- $a+(b-c)$
- $d^*(d+b)-a$
- $(a+b)^*(c-d)$
- $a+(b-c)^*d$

Leftmost and Rightmost Derivation

- ▶ Leftmost derivation
 - Leftmost nonterminal is replaced in each step
- ▶ Rightmost derivation
 - Rightmost nonterminal is replaced in each step
- ▶ Example
 - Grammar
 - > $S \rightarrow AB, A \rightarrow a, B \rightarrow b$
 - Leftmost derivation for "ab"
 - > $S \Rightarrow AB \Rightarrow aB \Rightarrow ab$
 - Rightmost derivation for "ab"
 - > $S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$

Parse Tree For Derivations

- ▶ Parse tree may be same for both leftmost & rightmost derivations

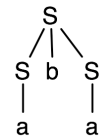
- Example Grammar: $S \rightarrow a \mid SbS$ String: aba

Leftmost Derivation

$S \Rightarrow SbS \Rightarrow abS \Rightarrow aba$

Rightmost Derivation

$S \Rightarrow SbS \Rightarrow Sba \Rightarrow aba$



- Parse trees don't show order productions are applied

- Every parse tree has a unique leftmost and a unique rightmost derivation

Parse Tree For Derivations (cont.)

- ▶ Not every string has a unique parse tree

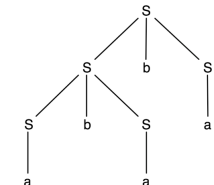
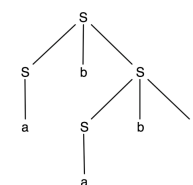
- Example Grammar: $S \rightarrow a \mid SbS$ String: $ababa$

Leftmost Derivation

$S \Rightarrow SbS \Rightarrow abS \Rightarrow abSbS \Rightarrow ababS \Rightarrow ababa$

Another Leftmost Derivation

$S \Rightarrow SbS \Rightarrow SbSbS \Rightarrow abSbS \Rightarrow ababS \Rightarrow ababa$

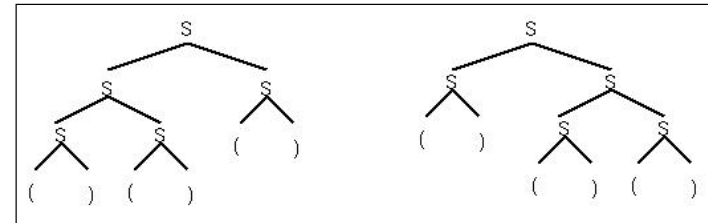


Ambiguity

- ▶ A grammar is **ambiguous** if a string may have multiple **leftmost** derivations
 - Equivalent to multiple parse trees
 - Can be hard to determine
- | | |
|---|------------|
| 1. $S \rightarrow aS \mid T$
$T \rightarrow bT \mid U$
$U \rightarrow cU \mid \epsilon$ | No |
| 2. $S \rightarrow T \mid T$
$T \rightarrow Tx \mid Tx \mid x \mid x$ | Yes |
| 3. $S \rightarrow SS \mid () \mid (S)$ | ? |

Ambiguity (cont.)

- ▶ Example
 - Grammar: $S \rightarrow SS \mid () \mid (S)$ String: $()()$
 - 2 distinct (leftmost) derivations (and parse trees)
 - > $S \Rightarrow \underline{SS} \Rightarrow \underline{SSS} \Rightarrow ()\underline{SS} \Rightarrow ()()\underline{S} \Rightarrow ()()()$
 - > $S \Rightarrow \underline{S}S \Rightarrow ()\underline{S} \Rightarrow ()\underline{SS} \Rightarrow ()()\underline{S} \Rightarrow ()()()$



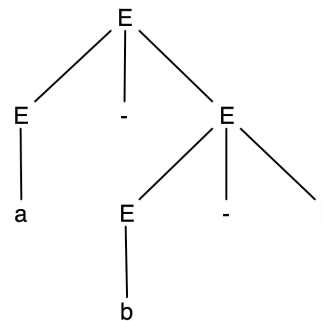
CFGs for Programming Languages

- ▶ Recall that our goal is to describe programming languages with CFGs
- ▶ We had the following example which describes limited arithmetic expressions

$$E \rightarrow a \mid b \mid c \mid E+E \mid E-E \mid E^*E \mid (E)$$
- ▶ What's wrong with using this grammar?
 - It's ambiguous!

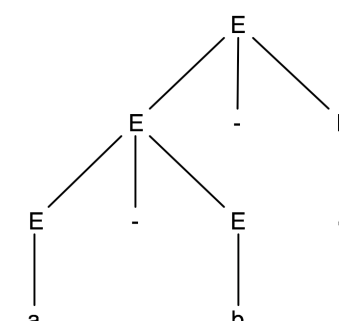
Example: a-b-c

$E \Rightarrow E-E \Rightarrow a-E \Rightarrow a-E-E \Rightarrow a-b-E \Rightarrow a-b-c$



Corresponds to $a-(b-c)$

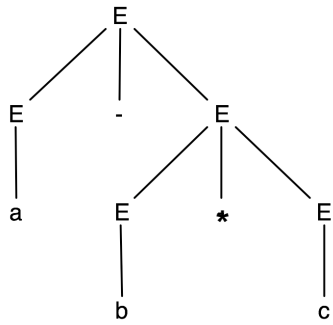
$E \Rightarrow E-E \Rightarrow E-E-E \Rightarrow a-E-E \Rightarrow a-b-E \Rightarrow a-b-c$



Corresponds to $(a-b)-c$

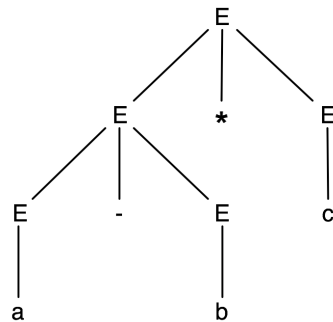
Example: $a-b^*c$

$E \Rightarrow E-E \Rightarrow a-E \Rightarrow a-E^*E \Rightarrow$
 $a-b^*E \Rightarrow a-b^*c$



Corresponds to $a-(b^*c)$

$E \Rightarrow E-E \Rightarrow E-E^*E \Rightarrow$
 $a-E^*E \Rightarrow a-b^*E \Rightarrow a-b^*c$



Corresponds to $(a-b)^*c$

Another Example: If-Then-Else

Aka **the dangling else problem**

$\langle \text{stmt} \rangle \rightarrow \langle \text{assignment} \rangle \mid \langle \text{if-stmt} \rangle \mid \dots$

$\langle \text{if-stmt} \rangle \rightarrow \text{if} (\langle \text{expr} \rangle) \langle \text{stmt} \rangle \mid$

$\text{if} (\langle \text{expr} \rangle) \langle \text{stmt} \rangle \text{ else } \langle \text{stmt} \rangle$

(Note $\langle \rangle$'s are used to denote nonterminals)

- Consider the following program fragment

$\text{if} (x > y)$

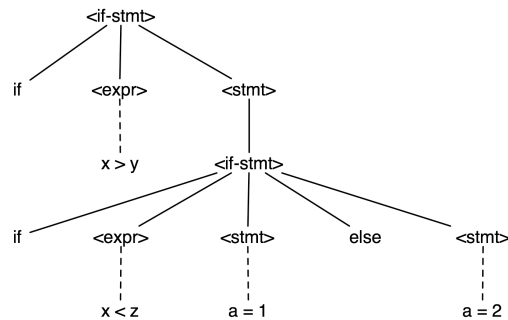
$\text{if} (x < z)$

$a = 1;$

$\text{else } a = 2;$

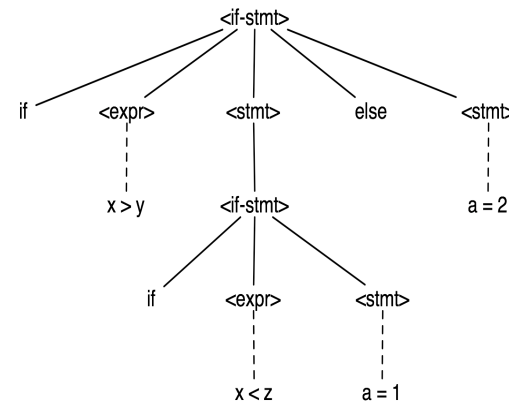
(Note: Ignore newlines)

Parse Tree #1



- Else belongs to inner if

Parse Tree #2



- Else belongs to outer if

Dealing With Ambiguous Grammars

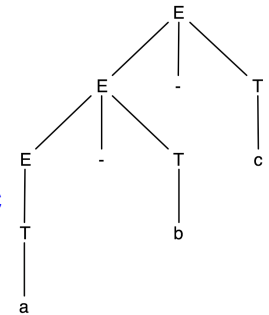
- ▶ Ambiguity is bad
 - Syntax is correct
 - But semantics differ depending on choice
 - ▶ Different associativity (a-b)-c vs. a-(b-c)
 - ▶ Different precedence (a-b)*c vs. a-(b*c)
 - ▶ Different control flow if (if else) vs. if (if) else
- ▶ Two approaches
 - Rewrite grammar
 - Use special parsing rules
 - ▶ Depending on parsing method (learn in CMSC 430)

CMSC 330

41

Fixing the Expression Grammar

- ▶ Require right operand to not be bare expression
 - $E \rightarrow E+T \mid E-T \mid E*T \mid T$
 - $T \rightarrow a \mid b \mid c \mid (E)$
- ▶ Corresponds to **left associativity**
- ▶ Now only one parse tree for **a-b-c**
 - Find derivation



CMSC 330

42

What If We Want Right Associativity?

- ▶ Left-recursive productions
 - Used for left-associative operators
 - Example
 - $E \rightarrow E+T \mid E-T \mid E*T \mid T$
 - $T \rightarrow a \mid b \mid c \mid (E)$
- ▶ Right-recursive productions
 - Used for right-associative operators
 - Example
 - $E \rightarrow T+E \mid T-E \mid T*E \mid T$
 - $T \rightarrow a \mid b \mid c \mid (E)$

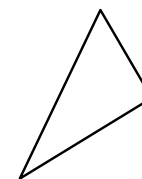
CMSC 330

43

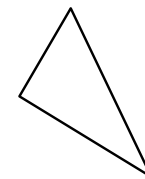
Parse Tree Shape

- ▶ The kind of recursion determines the shape of the parse tree

left recursion



right recursion



CMSC 330

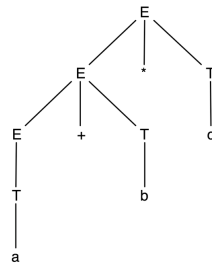
44

A Different Problem

- ▶ How about the string $a+b*c$?

$$E \rightarrow E+T \mid E-T \mid E*T \mid T$$

$$T \rightarrow a \mid b \mid c \mid (E)$$



- ▶ Doesn't have correct precedence for $*$

- When a nonterminal has productions for several operators, they effectively have the same precedence

- ▶ Solution – Introduce **new** nonterminals

CMSC 330

45

Final Expression Grammar

$$E \rightarrow E+T \mid E-T \mid T$$

lowest precedence operators

$$T \rightarrow T*P \mid P$$

higher precedence

$$P \rightarrow a \mid b \mid c \mid (E)$$

highest precedence (parentheses)

- ▶ Controlling precedence of operators

- Introduce new nonterminals
- Precedence increases closer to operands

- ▶ Controlling associativity of operators

- Introduce new nonterminals
- Assign associativity based on production form
 - ▶ $E \rightarrow E+T$ (left associative) vs. $E \rightarrow T+E$ (right associative)

CMSC 330

46

Tips For Designing Grammars

1. Use recursive productions to generate an arbitrary number of symbols

$$A \rightarrow xA \mid \epsilon \quad // \text{Zero or more } x' \text{ s}$$

$$A \rightarrow yA \mid y \quad // \text{One or more } y' \text{ s}$$

2. Use separate non-terminals to generate disjoint parts of a language, and then combine in a production

$$\{ a^*b^* \} \quad // a' \text{ s followed by } b' \text{ s}$$

$$S \rightarrow AB$$

$$A \rightarrow aA \mid \epsilon \quad // \text{Zero or more } a' \text{ s}$$

$$B \rightarrow bB \mid \epsilon \quad // \text{Zero or more } b' \text{ s}$$

CMSC 330

47

Tips For Designing Grammars (cont.)

3. To generate languages with matching, balanced, or related numbers of symbols, write productions which generate strings from the middle

$$\{ a^n b^n \mid n \geq 0 \} \quad // N a' \text{ s followed by } N b' \text{ s}$$

$$S \rightarrow aSb \mid \epsilon$$

$$\text{Example derivation: } S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

$$\{ a^n b^{2n} \mid n \geq 0 \} \quad // N a' \text{ s followed by } 2N b' \text{ s}$$

$$S \rightarrow aSbb \mid \epsilon$$

$$\text{Example derivation: } S \Rightarrow aSbb \Rightarrow aaSbbbb \Rightarrow aabbbb$$

CMSC 330

48

Tips For Designing Grammars (cont.)

4. For a language that is the union of other languages, use separate nonterminals for each part of the union and then combine

$\{ a^n(b^m|c^m) \mid m > n \geq 0 \}$

Can be rewritten as

$\{ a^n b^m \mid m > n \geq 0 \} \cup \{ a^n c^m \mid m > n \geq 0 \}$

$S \rightarrow T \mid V$

$T \rightarrow aTb \mid U$

$U \rightarrow Ub \mid b$

$V \rightarrow aVc \mid W$

$W \rightarrow Wc \mid c$