

CMSC 330: Organization of Programming Languages

Context Free Grammars 2

Last Lecture

- ▶ Why should we study CFGs?
 - Precisely describe syntax of programming languages
- ▶ What are the four parts of a CFG?
 - Terminals, nonterminals, productions, start symbol
- ▶ How do we tell if a string is accepted by a CFG?
 - Find a derivation from start symbol to string
 - > By applying productions to nonterminals at each step
- ▶ What's a parse tree?
 - Representation of derivation of string

CMSC 330

2

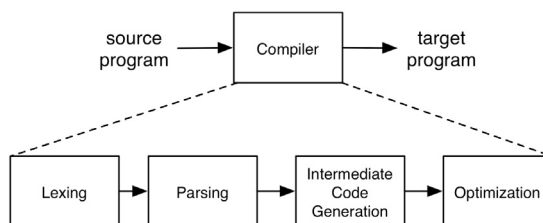
REs and CFGs in Practice

- ▶ REs turn raw text into a stream of **tokens**
 - E.g., "if", "then", "identifier", etc.
 - This process is called **scanning** or **lexing**
 - Whitespace and comments are simply skipped
 - These tokens become the input for the parser
- ▶ CFGs turn tokens into parse trees
 - This process is called **parsing**
 - Parse trees become the input for the code generator

CMSC 330

3

Steps of Compilation



CMSC 330

4

Leftmost and Rightmost Derivation

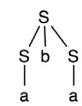
- ▶ Leftmost derivation
 - Leftmost nonterminal is replaced in each step
- ▶ Rightmost derivation
 - Rightmost nonterminal is replaced in each step
- ▶ Example
 - Grammar
 - > $S \rightarrow AB, A \rightarrow a, B \rightarrow b$
 - Leftmost derivation for "ab"
 - > $S \Rightarrow AB \Rightarrow aB \Rightarrow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

CMSC 330

5

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



CMSC 330

6

Parse Tree For Derivations (cont.)

- Not every string has a unique parse tree

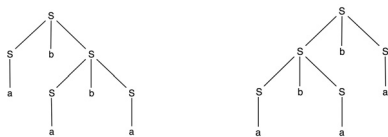
• Example Grammar: $S \rightarrow a \mid SbS$ String: **ababa**

Leftmost Derivation

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

Another Leftmost Derivation

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



CMSC 330

7

Ambiguity

- A grammar is **ambiguous** if a string may have multiple leftmost (or rightmost) derivations

- Equivalent to multiple parse trees
- Can be hard to determine

- $S \rightarrow aS \mid T$
 $T \rightarrow bT \mid U$ **No**
 $U \rightarrow cU \mid \epsilon$
- $S \rightarrow T \mid T$ **Yes**
 $T \rightarrow Tx \mid Tx \mid x \mid x$
- $S \rightarrow SS \mid () \mid (S)$ **?**

CMSC 330

8

Ambiguity (cont.)

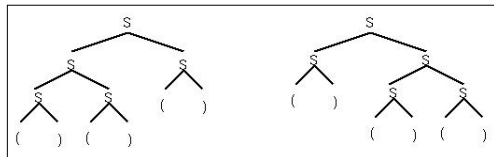
- Example

• Grammar: $S \rightarrow SS \mid () \mid (S)$ String: **()()**

• 2 distinct leftmost derivations (and parse trees)

> $S \Rightarrow SS \Rightarrow SSS \Rightarrow ()SS \Rightarrow ()()S \Rightarrow ()()()$

> $S \Rightarrow SS \Rightarrow (S)S \Rightarrow ()SS \Rightarrow ()()S \Rightarrow ()()()$



CMSC 330

9

More Derivations

- Is the following derivation leftmost or rightmost?

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

> Both! At most one non-terminal in each sentential form, so there's no choice in which non-terminals to expand

- How about the following derivation?

• Grammar: $S \rightarrow a \mid SbS$ String: **ababa**

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

> Neither! Selects left, center, left, and rightmost nonterminals

CMSC 330

10

Tips for Designing Grammars

- Use recursive productions to generate an arbitrary number of symbols
 - $A \rightarrow xA \mid \epsilon$ // Zero or more x's
 - $A \rightarrow yA \mid y$ // One or more y's
- Use separate non-terminals to generate disjoint parts of a language, and then combine in a production

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

$S \rightarrow AB$

$A \rightarrow aA \mid \epsilon$ // Zero or more a's

$B \rightarrow bB \mid \epsilon$ // Zero or more b's

CMSC 330

11

Tips for Designing Grammars (cont.)

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

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

$S \rightarrow aSb \mid \epsilon$

Example derivation: $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

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

$S \rightarrow aSbb \mid \epsilon$

Example derivation: $S \Rightarrow aSbb \Rightarrow aaSbbbb \Rightarrow aabbbb$

CMSC 330

12

Tips for Designing Grammars (cont.)

- 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

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

$S \rightarrow T \mid V$

$T \rightarrow aTb \mid U$

$U \rightarrow Ub \mid b$

$V \rightarrow aVc \mid W$

$W \rightarrow Wc \mid c$

CMSC 330

13

CFGs for 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?

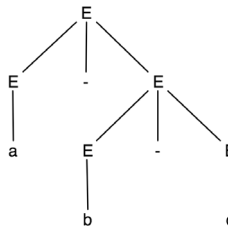
- It's ambiguous!

CMSC 330

14

Example: a-b-c

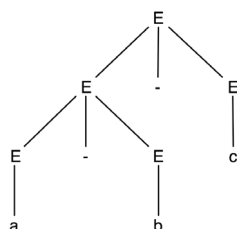
$E \Rightarrow E-E \Rightarrow a-E \Rightarrow a-E-E \Rightarrow a-b-E \Rightarrow a-b-c$



Corresponds to a-(b-c)

CMSC 330

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

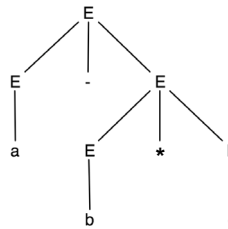


Corresponds to (a-b)-c

15

Example: a-b*c

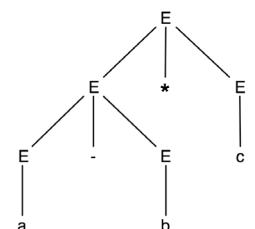
$E \Rightarrow E-E \Rightarrow a-E \Rightarrow a-E^*E \Rightarrow a-b^*E \Rightarrow a-b^*c$



Corresponds to a-(b*c)

CMSC 330

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



Corresponds to (a-b)*c

16

Another Example: If-Then-Else

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

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

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

(Note <>'s are used to denote nonterminals)

- Consider the following program fragment

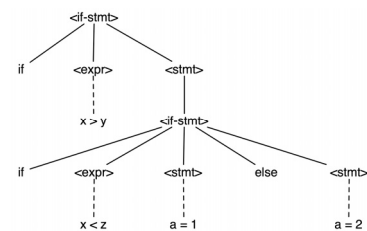
```
if (x > y)
  if (x < z)
    a = 1;
  else a = 2;
```

(Note: Ignore newlines)

CMSC 330

17

Parse Tree #1

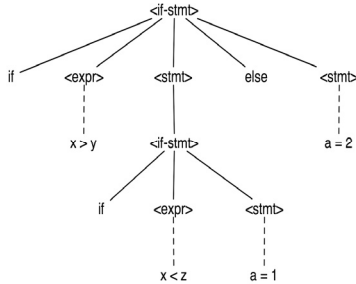


- Else belongs to inner if

CMSC 330

18

Parse Tree #2



- ▶ Else belongs to outer if

CMSC 330

19

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

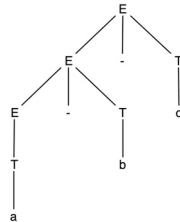
20

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

21

What if We Wanted 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
 - $E \rightarrow T+E \mid T-E \mid T^*E \mid T$
 - $T \rightarrow a \mid b \mid c \mid (E)$

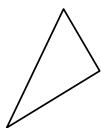
CMSC 330

22

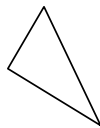
Parse Tree Shape

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

left recursion



right recursion



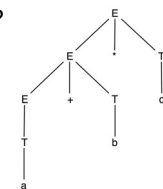
CMSC 330

23

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

CMSC 330

24

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)

► Practice

- Construct tree and left and right derivations for
 > $a+b*c$ $a*(b+c)$ $a*b+c$ $a-b-c$
- See what happens if you change the last set of productions to $P \rightarrow a \mid b \mid c \mid E \mid (E)$
- See what happens if you change the first set of productions to $E \rightarrow E+T \mid E-T \mid T \mid P$

CMSC 330

25

Summary

► Context free grammars

- Leftmost & rightmost derivations
- Ambiguity
- Designing grammars
- Associativity & precedence

CMSC 330

26