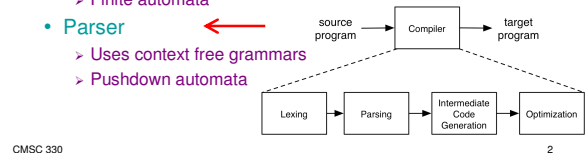


CMSC 330: Organization of Programming Languages

Context Free Grammars

Where We Are

- ▶ Programming languages
 - Ruby
 - OCaml
- ▶ Implementing programming languages
 - Scanner
 - Uses regular expressions
 - Finite automata
 - Parser
 - Uses context free grammars
 - Pushdown automata



CMSC 330

2

This Lecture

- ▶ Program syntax
 - Regular expressions & grammars
- ▶ Context free grammars (CFGs)
 - Definition
 - Derivations (leftmost)
 - Parse tree
 - Ambiguity
 - Associativity & precedence

CMSC 330

3

Programming Languages

- ▶ Syntax
 - What a program looks like
 - Needs to be very precise
- ▶ Semantics
 - What a program means
 - How different parts of a program behave

CMSC 330

4

Motivation for Grammars

- ▶ Programs are just strings of text
 - But they're strings that have a certain structure
- ▶ Informal description of structure of a C program
 - A C program is a list of declarations and definitions
 - A function definition contains parameters and a body
 - A function body is a sequence of statements
 - A statement is either an expression, a while loop, etc...
 - An expression may be assignment, addition, etc...

CMSC 330

5

Motivation for Grammars (cont.)

- ▶ We want to describe program structure precisely
- ▶ Regular expressions are not enough
 - No regular expression for balanced pairs of ()'s
 - { "(", ")", "((()))", "(((())))", ... } is not a regular language
- ▶ Need to use context free grammar (CFG)

CMSC 330

6

Context Free Grammar (CFG)

- ▶ A way of generating sets of strings or languages
- ▶ Grammar: $S \rightarrow 0S \mid 1S \mid \epsilon$
 - Means every S may be replaced by $0S$, $1S$, or ϵ
 - Example
 - > $S \Rightarrow 0S$ // using $S \rightarrow 0S$
 - $\Rightarrow 01S$ // using $S \rightarrow 1S$
 - $\Rightarrow 011S$ // using $S \rightarrow 1S$
 - $\Rightarrow 011$ // using $S \rightarrow \epsilon$
- ▶ Grammar is same as regular expression $(0|1)^*$
 - Generates / accepts the same set of strings

CMSC 330

7

Context-Free Grammars (CFGs)

- ▶ But CFGs can do a lot more!
 - $S \rightarrow (S) \mid \epsilon$ // generates 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

CMSC 330

8

Formal Definition

- ▶ A context-free grammar G is a 4-tuple
 - Σ – a finite set of terminal (alphabet) symbols
 - > 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 this means that the nonterminal can be replaced by the string of zero or more terminals or nonterminals to the right of the $\rightarrow$
 - > Can think of productions as rewriting rules
 - $S \in N$ – the start symbol

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

Informal Definition of Acceptance

- ▶ A string is **accepted** by a CFG if there is
 - Some sequence of applying productions (**rewrites**) starting at the start symbol that generates the string
- ▶ Example
 - Grammar: $S \rightarrow 0S \mid 1S \mid \epsilon$
 - Sequence generating the string 010
 - > $S \Rightarrow 0S \Rightarrow 01S \Rightarrow 010S \Rightarrow 010$
- ▶ Terminology
 - Such a sequence of rewrites is a **derivation** or **parse**
 - Discovering the derivation is called **parsing**

CMSC 330

11

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
 - $S \Rightarrow 0S \mid 1S \mid \epsilon$
- ▶ For the string 010
 - $S \Rightarrow 0S \Rightarrow 01S \Rightarrow 010S \Rightarrow 010$
 - $S \Rightarrow^+ 010$
 - $S \Rightarrow^* S$

CMSC 330

12

Practice

- Try to make a grammar which accepts
 - 0^*1^* — 0^n1^n where $n \geq 0$ — 0^n1^m where $m \leq n$
- $$\begin{aligned}
 S &\rightarrow A \mid B \\
 A &\rightarrow 0A \mid \epsilon \\
 B &\rightarrow 1B \mid \epsilon
 \end{aligned}$$
- Give some example strings from this language
 - $S \rightarrow 0 \mid 1S$
 - $> 0, 10, 110, 1110, 11110, \dots$
 - What language is it?
 - $> 1^*0$

CMSC 330

13

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.

CMSC 330

14

Formal Description of Example

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

CMSC 330

15

Uniqueness of Grammars

- Grammars are not unique
- Different grammars
 - Can generate the same set of strings (language)
- The following grammar generates the same set of strings as the previous grammar

$$\begin{aligned}
 E &\rightarrow E+T \mid E-T \mid T \\
 T &\rightarrow T^*P \mid P \\
 P &\rightarrow (E) \mid a \mid b \mid c
 \end{aligned}$$

CMSC 330

16

Notational Shortcuts

- A production is of the form
 - left-hand side (LHS) $\rightarrow$ right hand side (RHS)
- If not specified
 - Assume LHS of first listed 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
$A \rightarrow aA$	
$ \quad b$	// $A \rightarrow b$
$ $	// $A \rightarrow \epsilon$

CMSC 330

17

Sentential Forms

- Definition
 - A **sentential form** is a string of terminals and nonterminals produced from the start symbol
- Inductively
 - The start symbol
 - If $\alpha A \delta$ is a sentential form for a grammar, where $(\alpha \text{ and } \delta \in (N \cup \Sigma)^*)$, and $A \rightarrow \gamma$ is a production, then $\alpha \gamma \delta$ is a sentential form for the grammar
 - In this case, we say that $\alpha A \delta$ derives $\alpha \gamma \delta$ in one step, which is written as $\alpha A \delta \Rightarrow \alpha \gamma \delta$

CMSC 330

18

Sentential Forms Example

- ▶ Given grammar
 - $S \rightarrow 0S \mid 1S \mid \epsilon$
- ▶ Possible derivations
 - $S \Rightarrow 0S \Rightarrow 01S \Rightarrow 010S \Rightarrow 010$
 - > $S, 0S, 01S, 010S, 010$ are sentential forms
 - $S \Rightarrow 1S \Rightarrow 11S \Rightarrow 111S \Rightarrow 111$
 - > $S, 1S, 11S, 111S, 111$ are sentential forms
 - $S \Rightarrow \epsilon$
 - > S, ϵ are sentential forms
- ▶ In other words
 - If $S \Rightarrow^* \alpha$, then α is a sentential form

CMSC 330

19

Language Generated by Grammar

- ▶ A slightly more formal definition...
 - The language defined by a CFG is the set of all sentential forms made up of only terminals
- ▶ Example
 - $S \rightarrow 0S \mid 1S \mid \epsilon$
 - In language
 - $01, 000, 11, \epsilon \dots$
 - Not in language
 - $0S, a, 11S, \dots$

CMSC 330

20

Language Generated by Grammar (cont.)

- ▶ $L(G)$, the language generated by grammar G is

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

- S is the start symbol of the grammar
- Σ is the alphabet for that grammar
- ▶ In other words
 - All sentential forms with only terminals
 - All strings over Σ that can be derived from the start symbol via one or more productions

CMSC 330

21

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

22

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

23

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 (including ϵ)
- ▶ Reading the leaves left to right
 - Shows the string corresponding to the tree

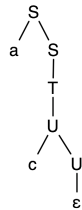
CMSC 330

24

Parse Tree Example

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

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



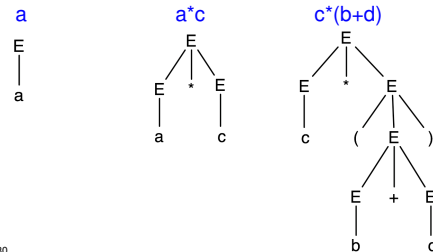
CMSC 330

25

Parse Trees for Expressions

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

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



CMSC 330

26

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...

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

CMSC 330

27

Review

- 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

28

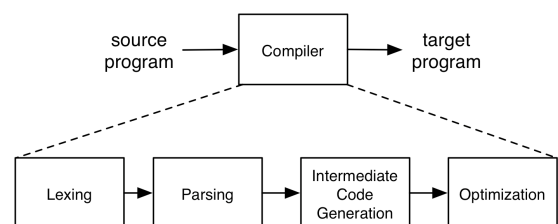
REs and CFGs in Practice

- 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

29

Steps of Compilation



CMSC 330

30

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

31

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

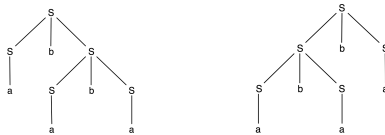
CMSC 330

32

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

33

Ambiguity

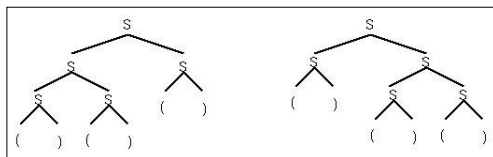
- A grammar is **ambiguous** if a string may have multiple leftmost (or rightmost) derivations
 - Equivalent to multiple parse trees
 - Can be hard to determine
 - $S \rightarrow aS \mid T$
 $T \rightarrow bT \mid U$
 $U \rightarrow cU \mid \epsilon$
 No
 - $S \rightarrow T \mid T$
 $T \rightarrow Tx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

CMSC 330

34

Ambiguity (cont.)

- Example
 - Grammar: $S \rightarrow SS \mid () \mid (S)$ String: **()()**
 - 2 distinct leftmost derivations (and parse trees)
 - $S \Rightarrow SS \Rightarrow SSS \Rightarrow ()SS \Rightarrow ()()S \Rightarrow ()()()$
 - $S \Rightarrow SS \Rightarrow ()S \Rightarrow ()SS \Rightarrow ()()S \Rightarrow ()()()$



CMSC 330

35

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

36

CFGs for Languages

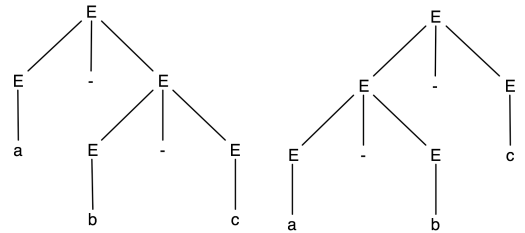
- Recall that our goal is to describe programming languages with CFGs
- We had the following example which describes limited arithmetic expressions
 $E \rightarrow a \mid b \mid c \mid E+E \mid E-E \mid E^*E \mid (E)$
 - It's ambiguous!
- What's wrong with using this grammar?

CMSC 330

37

Example: a-b-c

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



Corresponds to a-(b-c)

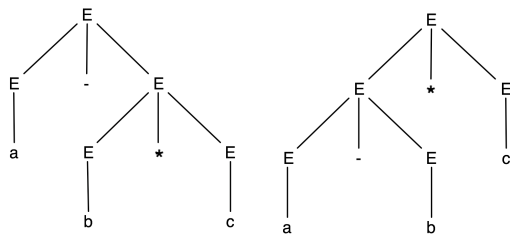
Corresponds to (a-b)-c

CMSC 330

38

Example: a-b*c

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



Corresponds to a-(b*c)

Corresponds to (a-b)*c

CMSC 330

39

Another Example: If-Then-Else

$\langle \text{stmt} \rangle \rightarrow \langle \text{assignment} \rangle \mid \langle \text{if-stmt} \rangle \mid \dots$
 $\langle \text{if-stmt} \rangle \rightarrow \text{if } (\langle \text{expr} \rangle) \langle \text{stmt} \rangle \mid$
 $\quad \text{if } (\langle \text{expr} \rangle) \langle \text{stmt} \rangle \text{ else } \langle \text{stmt} \rangle$
 (Note <>'s are used to denote nonterminals)

- Consider the following program fragment


```

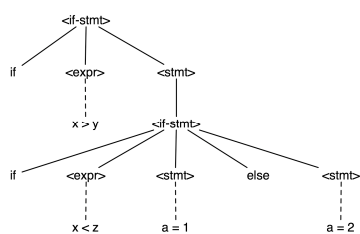
if (x > y)
  if (x < z)
    a = 1;
  else a = 2;
      
```

 (Note: Ignore newlines)

CMSC 330

40

Parse Tree #1

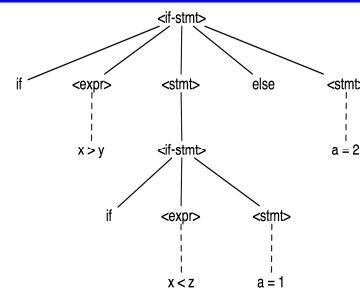


- Else belongs to inner if

CMSC 330

41

Parse Tree #2



- Else belongs to outer if

CMSC 330

42

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

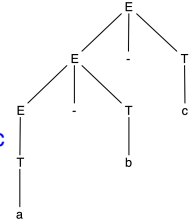
43

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

44

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

45

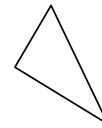
Parse Tree Shape

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

left recursion



right recursion



CMSC 330

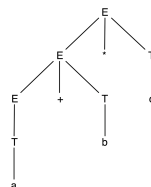
46

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

47

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

48

Tips for Designing Grammars

1. 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
2. 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

49

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
 $\{a^n b^n \mid n \geq 0\}$ // 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

50

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
 $\{a^n b^m \mid m > n \geq 0\} \cup \{a^n c^m \mid m > n \geq 0\}$
 $S \rightarrow T \mid V$
 $T \rightarrow aTb \mid U$
 $U \rightarrow Ub \mid b$
 $V \rightarrow aVc \mid W$
 $W \rightarrow Wc \mid c$

CMSC 330

51