

# CMSC 330: Organization of Programming Languages

---

## Context Free Grammars

### This Lecture

---

- ▶ Program syntax
  - Regular expressions & grammars
- ▶ Context free grammars (CFGs)
  - Definition
  - Languages accepted
  - Derivations
  - Sentential forms
  - Parse tree

CMSC 330

3

### Last Lecture

---

- ▶ Reducing NFA to DFA
  - $\epsilon$ -closure
  - Subset algorithm
- ▶ Minimizing DFA
  - Moore reduction
- ▶ Complementing DFA
- ▶ Implementing DFA

CMSC 330

2

### 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  $\epsilon$
  - 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
  - $\Sigma$  – 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  $m \leq n$
- $$\begin{aligned}
 S &\rightarrow A \mid B \\
 A &\rightarrow 0A \mid \epsilon \\
 B &\rightarrow 1B \mid \epsilon
 \end{aligned}$$
- $$\begin{aligned}
 S &\rightarrow 0S1 \mid \epsilon \\
 S &\rightarrow 0S1 \mid 0S \epsilon
 \end{aligned}$$
- ▶ Give some example strings from this language
    - $S \rightarrow 0 \mid 1S$ 
      - > 0, 10, 110, 1110, 11110, ...
    - 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
  - $E \rightarrow E+T \mid E-T \mid T$
  - $T \rightarrow T^*P \mid P$
  - $P \rightarrow (E) \mid a \mid b \mid c$

CMSC 330

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  $\epsilon$

$S \rightarrow ABC$	// S is start symbol
$A \rightarrow aA$	
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  $\alpha$  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,  $\epsilon$  are sentential forms
- ▶ In other words
  - If  $S \Rightarrow^* \alpha$ , then  $\alpha$  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,  $\epsilon$  ...
  - Not in language
    - 0S, a, 11S, ...

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
- $\Sigma$  is the alphabet for that grammar
- ▶ In other words
  - All sentential forms with only terminals
  - All strings over  $\Sigma$  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  $\epsilon$ )
- ▶ Reading the leaves left to right
  - Shows the string corresponding to the tree

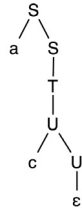
CMSC 330

24

## Parse Tree Example

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

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



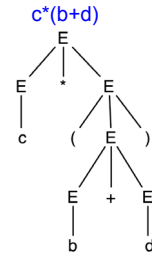
CMSC 330

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