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

REs and CFGs in Practice

- REs turn raw text into a stream of tokens
  - E.g., “if”, “then”, “identifier”, etc.
  - This process is calling 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

Steps of Compilation

- Parse tree may be same for both leftmost & rightmost 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

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 (cont.)

- Not every string has a unique parse tree
  - Example Grammar: $S \rightarrow a | SbS$
    - String: $ababa$
        - Leftmost Derivation:
          $$S \Rightarrow SbS \Rightarrow abSbS \Rightarrow ababS \Rightarrow ababa$$
        - 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 (or rightmost) derivations
  - Equivalent to multiple parse trees
  - Can be hard to determine
    1. $S \rightarrow aS | T$
    2. $S \rightarrow T | T$
    3. $S \rightarrow SS | () | (S)$

Tips for Designing Grammars

1. Use recursive productions to generate an arbitrary number of symbols
   - $A \rightarrow xA | \varepsilon$
   - $A \rightarrow yA | y$
2. Use separate non-terminals to generate disjoint parts of a language, and then combine in a production
   - $\{a^nb^n | n \geq 0\}$
   - $\{a^n | n \geq 0\}$
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

\[ \{ a^n b^m c^n | m > n \geq 0 \} \]

Can be rewritten as

\[ \{ a^n b^m | m > n \geq 0 \} \cup \{ a^n c^n | m > n \geq 0 \} \]

S → T | V
T → aTb | U
U → Ub | b
V → aVc | W
W → Wc | c

CFGs for Languages

Recall that our goal is to describe programming languages with CFGs

We had the following example which describes limited arithmetic expressions

\[ E \rightarrow a | b | c | E+E | E-E | E*E | (E) \]

What’s wrong with using this grammar?

• It’s ambiguous!

Example: a-b-c

\[ E \rightarrow E-E \rightarrow a-E \rightarrow a-b-E \rightarrow a-b-c \]

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)

\[ E \rightarrow E-E \rightarrow E*E \rightarrow a-E*E \rightarrow a-b*E \rightarrow a-b*c \]

Corresponds to (a-b)*c

Another Example: If-Then-Else

\[ \langle stmt \rangle \rightarrow \langle assignment \rangle \mid \langle if-stmt \rangle \mid \ldots \]

\[ \langle if-stmt \rangle \rightarrow \text{if (expr) stmt} \mid \text{if (expr) stmt} \text{ else stmt} \]

(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)

Parse Tree #1

Else belongs to inner 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)

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

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
    \[ E \rightarrow T+E \mid T-E \mid T*E \mid T \]
    \[ T \rightarrow a \mid b \mid c \mid (E) \]

Parse Tree Shape

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

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
Final Expression Grammar

- $E \rightarrow E + T | E - T | T$
- $T \rightarrow T^* P | P$
- $P \rightarrow a | b | c | (E)$

- lowest precedence operators
- higher precedence
- 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 $P \rightarrow a | b | c | E | (E)$
- See what happens if you change the first set of productions to $E \rightarrow E + T | E - T | T | P$

Summary

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