CMSC 330: Organization of Programming Languages

Context-Free Grammars

Review

- Why should we study CFGs?
- What are the four parts of a CFG?
- How do we tell if a string is accepted by a CFG?
- What’s a parse tree?
Review

A *sentential form* is a string of terminals and non-terminals produced from the start symbol

Inductively:
- The start symbol
- If \( \alpha A \delta \) is a sentential form for a grammar, where \( \alpha \) and \( \delta \in (N|\Sigma)^* \), and \( A \to \gamma \) 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 \( \alpha A \delta \Rightarrow \alpha \gamma \delta \)

Leftmost and Rightmost Derivation

- Example: \( S \to a | SbS \)  
  - String: aba

  **Leftmost Derivation**
  \[
  S \Rightarrow SbS \Rightarrow abS \Rightarrow aba
  \]
  At every step, apply production to leftmost non-terminal

  **Rightmost Derivation**
  \[
  S \Rightarrow SbS \Rightarrow Sba \Rightarrow aba
  \]
  At every step, apply production to rightmost non-terminal

- Both derivations happen to have the same parse tree
- A parse tree has a unique leftmost and a unique rightmost derivation
- Not every string has a unique parse tree
- Parse trees don’t show the order productions are applied
Another Example (cont’d)

S → a | SbS

- Is ababa in this language?

A leftmost derivation

S ⇒ SbS ⇒ abS ⇒
   abSbS ⇒ ababS ⇒ ababa

Another leftmost derivation

S ⇒ SbS ⇒ SbSbS ⇒
   abSbS ⇒ ababS ⇒ ababa

Ambiguity

- A string is ambiguous for a grammar if it has more than one parse tree
  - Equivalent to more than one leftmost (or more than one rightmost) derivation

- A grammar is ambiguous if it generates an ambiguous string
  - It’s can be hard to see this with manual inspection

- Exercise: can you create an unambiguous grammar for S → a | SbS?
Are these Grammars Ambiguous?

(1) \[ S \rightarrow aS \mid T \]
\[ T \rightarrow bT \mid U \]
\[ U \rightarrow cU \mid \epsilon \]

(2) \[ S \rightarrow T \mid T \]
\[ T \rightarrow Tx \mid Tx \mid x \mid x \]

(3) \[ S \rightarrow SS \mid () \mid (S) \]

Ambiguity of Grammar (Example 3)

- 2 different parse trees for the same string: 
  \(()()()\)
- 2 distinct leftmost derivations:
  \[ S \Rightarrow SS \Rightarrow SS \Rightarrow ()SS \Rightarrow ()()S \Rightarrow ()()() \]
  \[ S \Rightarrow SS \Rightarrow ()S \Rightarrow ()SS \Rightarrow ()()S \Rightarrow ()()() \]

- We need unambiguous grammars to manage programming language semantics
More on Leftmost/Rightmost Derivations

- Is the following derivation leftmost or rightmost?
  \[ S \Rightarrow aS \Rightarrow aT \Rightarrow aU \Rightarrow acU \Rightarrow ac \]
  - There’s at most one non-terminal in each sentential form, so there’s no choice between left or right non-terminals to expand

- How about the following derivation?
  - \[ S \Rightarrow SbS \Rightarrow SbSbS \Rightarrow SbabS \Rightarrow ababS \Rightarrow ababa \]

Tips for Designing Grammars

1. Use recursive productions to generate an arbitrary number of symbols
   \[ A \rightarrow xA \ | \ \varepsilon \]  Zero or more x’s
   \[ A \rightarrow yA \ | \ y \]  One or more y’s

2. Use separate non-terminals to generate disjoint parts of a language, and then combine in a production
   \[ G = S \rightarrow AB \]
   \[ A \rightarrow aA \ | \ \varepsilon \]
   \[ B \rightarrow bB \ | \ \varepsilon \]
   \[ L(G) = a^*b^* \]
Tips for Designing Grammars (cont’d)

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

\{a^n b^n | n \geq 0\} \text{ (not a regular language!)}
S \to aSb | \varepsilon

Example: S \to aSb \Rightarrow aaSbb \Rightarrow aabb

\{a^n b^{2n} | n \geq 0\}
S \to aSbb | \varepsilon

Tips for Designing Grammars (cont’d)

\{a^n b^m | m \geq 2n, n \geq 0\}
S \to aSbb | B | \varepsilon
B \to bB | b

The following grammar also works:
S \to aSbb | B
B \to bB | \varepsilon

How about the following?
S \to aSbb | bS | \varepsilon
Tips for Designing Grammars (cont’d)

\{a^n b^m a^{n+m} \mid n \geq 0, m \geq 0\}
Rewrite as \(a^n b^m a^m a^n\), which now has matching superscripts (two pairs)

Would this grammar work?
\[S \rightarrow aSa \mid B\]
\[B \rightarrow bBa \mid ba\]

Corrected:
\[S \rightarrow aSa \mid B\] The outer \(a^n a^n\) are generated first,
\[B \rightarrow bBa \mid \varepsilon\] then the inner \(b^m a^m\)

Tips for Designing Grammars (cont’d)

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

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

Can be rewritten as
\[\{ a^n b^m \mid m > n \geq 0\} \cup \{ a^n c^m \mid m > n \geq 0\}\]
Tips for Designing Grammars (cont’d)

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

S \rightarrow T \mid U
T \rightarrow aTb \mid Tb \mid b \quad \text{T generates the first set}
U \rightarrow aUc \mid Uc \mid c \quad \text{U generates the second set}

• What’s the parse tree for string \text{abbb}?
  • Ambiguous!

![Parse Tree Diagram]

Tips for Designing Grammars (cont’d)

Will this fix the ambiguity?

S \rightarrow T \mid U
T \rightarrow aTb \mid bT \mid b
U \rightarrow aUc \mid cU \mid c

• It’s not ambiguous, but it can generate invalid strings such as \text{babb}
Tips for Designing Grammars (cont’d)

\{ a^nb^m | m > n \geq 0 \} \cup \{ a^nc^m | m > n \geq 0 \}

Unambiguous version

\begin{align*}
S & \rightarrow T \ | \ V \\
T & \rightarrow aTb \ | \ U \\
U & \rightarrow Ub \ | \ b \\
V & \rightarrow aVc \ | \ W \\
W & \rightarrow Wc \ | \ c
\end{align*}

CFGs for Languages

• Recall that our goal is to describe programming languages with CFGs

• We had the following example which describes limited arithmetic expressions

\begin{align*}
E & \rightarrow a \ | \ b \ | \ c \ | \ E+E \ | \ E-E \ | \ E*E \ | \ (E)
\end{align*}

• What’s wrong with using this grammar?
  – It’s ambiguous!
The Issue: Associativity

- Ambiguity is bad here because if the compiler needs to generate code for this expression, it doesn’t know what the programmer intended

- So what do we mean when we write a-b-c?
  - In mathematics, this only has one possible meaning
  - It’s (a-b)-c, since subtraction is left-associative
  - a-(b-c) would be the meaning if subtraction was right-associative
Another Example: If-Then-Else

<stmt> ::= <assignment> | <if-stmt> | ...
<if-stmt> ::= if (<expr>) <stmt> |
             if (<expr>) <stmt> else <stmt>
– (Here <>’s are used to denote nonterminals and ::= for productions)

• 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
Parse Tree #2

• Else belongs to outer if

Fixing the Expression Grammar

• Idea: Require that the right operand of all of the operators is not a bare expression
  
  E → E+T | E-T | E*T | T
  T → a | b | c | (E)

• Now there's only one parse tree for a-b-c

  – Exercise: Give a derivation for the string a-(b-c)
What if We Wanted Right-Associativity?

- Left-recursive productions are used for left-associative operators
- Right-recursive productions are used for right-associative operators

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

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

Parse Tree Shape

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

- Exercise: draw a parse tree for \(a-b-c\) in the prior grammar in which subtraction is right-associative
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

- How can we fix this?

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)

- Exercises:
  
  - Construct tree and left and right derivations for
    
    $a + b * c \quad a * (b + c) \quad a * b + c \quad a - b - c$
  
  - 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 $E \rightarrow E + T \mid E - T \mid T \mid P$