CMSC 330: Organization of Programming Languages

Context-Free Grammars

Motivation

- Programs are just strings of text
  - But they're strings that have a certain structure
    - 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, an if, a goto, etc.
    - An expression may be assignment, addition, subtraction, etc
- We want to solve two problems
  - We want to describe programming languages precisely
  - We need to describe more than the regular languages
    - Recall that regular expressions, DFAs, and NFAs are limited in their expressiveness

Context-Free Grammars (CFGs)

- A way of generating sets of strings or languages
- They subsume regular expressions (and DFAs and NFAs)
  - There is a CFG that generates any regular language
  - (But regular expressions are a better notation for languages that are regular.)
- They can be used to describe programming languages
  - They describe the parsing process (mostly)

Formal Definition

- A context-free grammar G is a 4-tuple:
  - Σ – a finite set of *terminal* or *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 $\rightarrow$
  - $S \in N$ – the *start symbol*

Example: Arithmetic Expressions (Limited)

- Σ = { +, -, *, (,), a, b, c }, N = { E }
  P = { E → a, E → b, E → c, E → E+E,
  E → E-E, E → E*E, E → (E)}, S = 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 + d),…}
- Example strings not in the language
  - d, c(a), a+, b**c, etc.

Notational Shortcuts

- If not specified, assume the left-hand side of the first listed production is the start symbol
- Usually productions with the same left-hand sides are combined with |
- If a production has an empty right-hand side it means ε
- Using these shortcuts we could instead write this grammar as
  E → a | b | c | E+E | E-E | E*E | (E)

Another Example Grammar

- S → aS | T
  T → bT | U
  U → cU | ε

Sentential Forms and Derivations

- A *sentential form* is a string of terminals and nonterminals produced from that grammar's start symbol
  - The start symbol is a sentential form for a grammar
  - If $\alpha A\beta$ is a sentential form for a grammar, where (α and $\beta \in (N|\Sigma)^*$), and $A \rightarrow \gamma$ is a production, then $\alpha\gamma\beta$ is a sentential form for the grammar
    - In this case, we say that $\alpha A\beta$ *derives* $\alpha\gamma\beta$ in one step, which is written as $\alpha A\beta \Rightarrow \alpha\gamma\beta$
- $\Rightarrow^+$ is used to indicate a derivation of one or more steps
- $\Rightarrow^*$ indicates a derivation of zero or more steps

Example

$S \rightarrow aS \mid T$

$T \rightarrow bT \mid U$

$U \rightarrow cU \mid \varepsilon$

- A derivation:
  - $S \Rightarrow aS \Rightarrow aT \Rightarrow aU \Rightarrow acU \Rightarrow ac$
    - Abbreviated as $S \Rightarrow^+ ac$
    - So S, aS, aT, aU, acU, ac are all sentential forms for this grammar
  - $S \Rightarrow T \Rightarrow U \Rightarrow \varepsilon$
- Is there any derivation
  - $S \Rightarrow^+ ccc$? $S \Rightarrow^+ Sa$?
  - $S \Rightarrow^+ bab$? $S \Rightarrow^+ bU$?

The Language Generated by a CFG

- The *language generated by a grammar G* is

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

  - (where S is the start symbol of the grammar and Σ is the alphabet for that grammar)
- I.e., all sentential forms with only terminals
- I.e., all strings over Σ that can be derived from the grammar's start symbol by applying one or more productions

Example (cont'd)

$S \rightarrow aS \mid T$

$T \rightarrow bT \mid U$

$U \rightarrow cU \mid \varepsilon$

- Generates what language?
- Do other grammars generate this language?

$S \rightarrow ABC$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

$C \rightarrow cC \mid \varepsilon$

  - So grammars are not unique

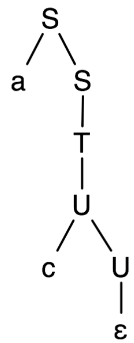
Parse Trees

- A *parse tree* represents a derivation:
  - The root node is the start symbol
  - Each interior node is a nonterminal
  - The children of a node are the symbols on the right-hand side of the production applied to that nonterminal
  - The leaves are all terminal symbols
- Reading the leaves left-to-right shows the string corresponding to the tree

Example

S ⇒ aS ⇒ aT ⇒ aU ⇒ acU ⇒ ac

S → aS | T
T → bT | U
U → cU | ε



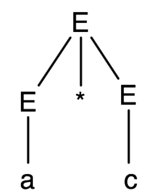
Parse Trees for Expressions

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

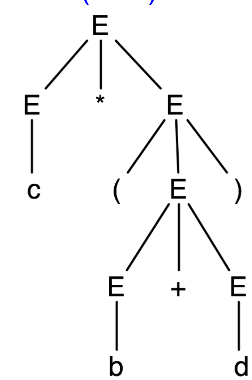
E → a | b | c | d | E+E | E-E | E*E | (E)

a

a*c



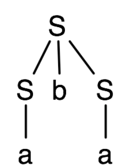
c*(b+d)



Another Example

S → a | SbS

- Is a in the language generated by this grammar?
- How about aba?
  - Yes, there are two possible derivations
    - S ⇒ SbS ⇒ abS ⇒ aba
      - This is a *leftmost derivation*
      - At every step, apply production to leftmost nonterminal
    - S ⇒ SbS ⇒ Sba ⇒ aba
      - This is a *rightmost* derivation
  - Both derivations have the same parse tree
    - A parse tree has a unique leftmost and a unique rightmost derivation
    - Parse trees don't show the order productions were applied



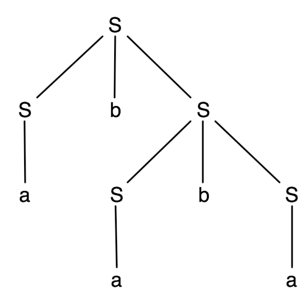
Another Example (cont'd)

S → a | SbS

- Is ababa in this language?

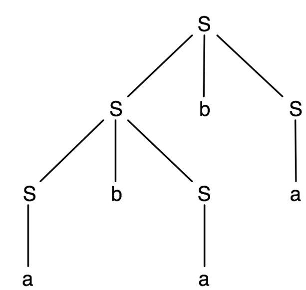
A leftmost derivation

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 can be hard to see this with manual inspection
- Exercise: can you create an unambiguous grammar which generates the same language?

Are these Grammars Ambiguous?

S → aS | T
T → bT | U
U → cU | ε

S → T | T
T → Tx | Tx | x | x

More on Leftmost/Rightmost Derivations

- Is the following derivation leftmost or rightmost?
  S ⇒ aS ⇒ aT ⇒ aU ⇒ acU ⇒ ac
  - There's at most one nonterminal in each sentential form, so there's no choice between left or right nonterminals to expand
- How about the following derivation?
  - S ⇒ SbS ⇒ SbSbS ⇒ SbabS ⇒ ababS ⇒ ababa

Tips for Designing Grammars

1. Use recursive productions to generate an arbitrary number of symbols
   A → xA | ε Zero or more x's
   A → yA | y One or more y's
2. Use separate nonterminals to generate disjoint parts of a language, and then combine in a production
   G = S → AB
   A → aA | ε
   B → bB | ε
   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

$\{a^n b^n \mid n \geq 0\}$ (not a regular language!)

$S \rightarrow aSb \mid \varepsilon$

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

$\{a^n b^{2n} \mid n \geq 1\}$

$S \rightarrow aSbb \mid abb$

Tips for Designing Grammars (cont'd)

$\{a^n b^m \mid m \geq 2n, n \geq 0\}$

$S \rightarrow aSbb \mid B \mid \varepsilon$

$B \rightarrow bB \mid b$

The following grammar also works:

$S \rightarrow aSbb \mid B$

$B \rightarrow bB \mid \varepsilon$

How about the following?

$S \rightarrow aSbb \mid bS \mid \varepsilon$

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:

$S \rightarrow aSa \mid B$

$B \rightarrow bBa \mid \varepsilon$

The outer $a^n a^n$ are generated first, then the inner $b^m a^m$