Recall: Context-Free Grammars

- Regular Expressions are great ... but not good enough to capture a programming language!
  - Example: RE for balanced pairs of parentheses
    - \( L = \{()\}, \left\{ (())\right\}, \ldots \) (same # of "(" and ")")

- CFGs subsume REs:
  - \((a | b)^* \) same as \( S \rightarrow aS | bS \)

Generating Strings from CFGs

- \( S \rightarrow aS | bS | \)
  - Means "we can replace \( S \) with \( aS \), \( bS \), or \( e \)"
  - Generate (derive) string "abba"
    - \( S \rightarrow aS \rightarrow abS \rightarrow abba \rightarrow abba \)

- \( S \rightarrow (S) | ( \)
  - Matching pairs of ( )
    - \( S \rightarrow (S) \rightarrow ((S)) \rightarrow ( ) \)

Describing Grammars

- Example 1: \( S \rightarrow abS | a \)
  - \((ab)^*a\)

- Example 2: \( S \rightarrow aSb | \)
  - Any # of \( a \)'s followed by the same number of \( b \)'s
    - \( a^n b^n \)
  - RE?

- Example 3: \( S \rightarrow aS | T, T \rightarrow bT | U, U \rightarrow cU | \)
  - Any # of \( a \)'s, followed by any # of \( b \)'s, followed by any # of \( c \)'s
    - \( a^* b^* c^* \)

Deriving Strings

- Example 1: \( S \rightarrow abS | a \)

- Example 2: \( S \rightarrow aSb | \)

- Example 3: \( S \rightarrow aS | T, T \rightarrow bT | U, U \rightarrow cU | \)
Deriving Strings

- Example 1: $S \rightarrow abS | a$
  - a
  - ababa

- Example 2: $S \rightarrow aSb |$
  - ab
  - aaabb

- Example 3: $S \rightarrow aS | T, T \rightarrow bT | U, U \rightarrow cU |$
  - aabbbcc
  - bbc

Working Toward PLs

- Basic Arithmetic Expressions

- Boolean Expressions

Properties of Grammars - Ambiguity

- Ambiguity
  - Multiple leftmost or rightmost derivations

- Leftmost/Rightmost Derivation
  - When deriving string, always derive ___-most non-terminal first

  - Example: $S \rightarrow S$ and $S | S or S | (S) | true | false$
  - Derive ((true and false) or (false and true and true))

Leftmost Derivation

- Leftmost:
  - $S \Rightarrow$
  - $(S) \Rightarrow$
  - $(S or S) \Rightarrow$
  - $(S and S) or S) \Rightarrow$
  - $(true and S) or S) \Rightarrow$
  - $(true and false) or S) \Rightarrow$
  - $(true and false) or (S)) \Rightarrow$
  - $(true and false) or (S and S)) \Rightarrow$
  - $(true and false) or (false and S and S)) \Rightarrow$
  - $(true and false) or (false and true and S)) \Rightarrow$
  - $(true and false) or (false and true and true))$

Properties of Grammars - Ambiguity

- Is our grammar for basic boolean expressions ambiguous?
  - If so, what is an example?
Properties of Grammars - Ambiguity

- Is our grammar for basic boolean expressions ambiguous?
  - Yes; Consider the leftmost derivation of "true and true and true":
    - $S \Rightarrow S$ and $S \Rightarrow true$ and $S \Rightarrow true$ and $S \Rightarrow true$
    - $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow true$ and $S \Rightarrow true$

Designing Grammars

- Tip 1: Use recursive productions to generate an arbitrary number of symbols:
  - $A \rightarrow aA | \epsilon$ (zero or more As)
  - $B \rightarrow bB | b$ (one or more Bs)
- Tip 2: Use separate nonterminals to consider disjoint parts of a language, then combine with a production:
  - $G \rightarrow AB$
  - $A \rightarrow aA$ (grammar for $a^*bb^*$)
  - $B \rightarrow bB | b$
- Tip 3: To generate languages with matching, balanced, or related numbers of symbols, write productions which generate strings from the middle:
  - $S \rightarrow aSb$ ($a^n b^n$)
  - What about $a^n b^{2n}$?
- Tip 4: For a language that’s a union of other languages, use separate non-terminals for each part of the union, then combine:
  - $(a^n (b^m | c^m), m > n >= 0)$
  - $(a^n b^m, m > n >= 0) U (a^n c^m, m > n >= 0)$

Designing Grammars

- What is the grammar for this expression?
  - $S \rightarrow T | U$
  - $T \rightarrow aTb | Tb | b$ (but it’s AMBIGUOUS!)
  - $U \rightarrow aUc | Uc | c$ (consider abbb)

- Fixing ambiguity is a later topic ... any thoughts?
Practice Designing Grammars

1. \( a^x b^y \), where \( x = y \)
2. \( a^x b^y \), where \( x > y \)
3. \( a^x b^y \), where \( x = 2y \)
4. \( a^x b^y \), where \( z = x + y \)
5. All strings of \( a \) and \( b \) that are palindromes.
6. All strings of \( a \) and \( b \) that include substring “baa”.
7. All strings of \( a \) and \( b \) with an odd number of \( a \)'s and an odd number of \( b \)'s.

Practice Designing Grammars

1. \( a^x b^y \), where \( x = y \)
   1. \( S \rightarrow aSb \) |
   2. \( aL \) |
   3. \( aLb \) |
2. \( a^x b^y \), where \( x > y \)
   1. \( S \rightarrow aL \)
   2. \( L \rightarrow aL \) | \( aLb \) |
3. \( a^x b^y \), where \( x = 2y \)
   1. \( S \rightarrow aaSb \)

Practice Designing Grammars

4. \( a^x b^y a^z \), where \( z = x + y \)
   1. \( S \rightarrow aSa | L \)
   2. \( L \rightarrow bLa | \)
5. All strings of \( a \) and \( b \) that are palindromes.
   1. \( S \rightarrow aSa | bSb | L \)
   2. \( L \rightarrow a | b | \)

Practice Designing Grammars

6. All strings of \( a \) and \( b \) that include substring “baa”.
   1. \( S \rightarrow LbaaL \)
   2. \( L \rightarrow aL | bL | \) (all strings over \( a,b \))
7. All strings of \( a \) and \( b \) with an odd number of \( a \)'s and an odd number of \( b \)'s.
   1. \( S \rightarrow EaEbE | EbEaE \)
   2. \( E \rightarrow EaEaE | EbEbE | SS | \) (even # of \( a \)'s & \( b \)'s)