Context Free Languages

Parse Trees and Ambiguity

Plan for 2nd half

- Ambiguous Grammars and Parse Trees
- Questions?

Text note

- We will not be covering the conversions / proofs .

Parse Trees

- Graphical means to illustrate a derivation of a string from a grammar
  - Root of the tree = start variable
  - Interior nodes = other variables
    - Children of nodes = application of a production rule
  - Leaf nodes = Terminal symbols

Another example

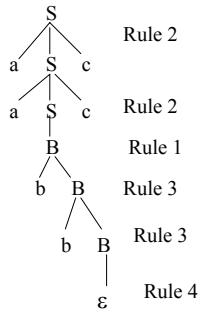
- Find a CFG to describe:
  - $L = \{a^i b^j c^k \mid i = k\}$
    - $S \rightarrow B$ (1)
    - $S \rightarrow aSc$ (2)
    - $B \rightarrow bB$ (3)
    - $B \rightarrow \varepsilon$ (4)
  - Can also write as
    - $S \rightarrow B \mid aSc$
    - $B \rightarrow bB \mid \varepsilon$

Another example

- Let's derive a string from L: aabbcc
  - $S \Rightarrow aSc$ rule 2
    $S \Rightarrow aaScc$ rule 2
    $S \Rightarrow aaBcc$ rule 1
    $S \Rightarrow aabBcc$ rule 3
    $S \Rightarrow aabbBcc$ rule 3
    $S \Rightarrow aabb \varepsilon cc$ rule 4
    $= aabbcc$

Parse Tree

- An inorder traversal of the tree will give the the string derived.



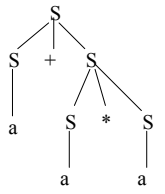
Recall our example from last time

- Defining the grammar for algebraic expressions – Production rules
  - $S \rightarrow S + S$ (1)
    $S \rightarrow S - S$ (2)
    $S \rightarrow S * S$ (3)
    $S \rightarrow S / S$ (4)
    $S \rightarrow (S)$ (5)
    $S \rightarrow a$ (6)

One more example

- Show derivation for a + a * a
  - $S \Rightarrow S + S$ rule 1
    $S \Rightarrow a + S$ rule 6
    $S \Rightarrow a + S * S$ rule 3
    $S \Rightarrow a + a * S$ rule 6
    $S \Rightarrow a + a * a$ rule 6

Parse Tree

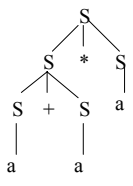


- $S \Rightarrow S + S$ rule 1
  $S \Rightarrow a + S$ rule 6
  $S \Rightarrow a + S * S$ rule 3
  $S \Rightarrow a + a * S$ rule 6
  $S \Rightarrow a + a * a$ rule 6

One more example

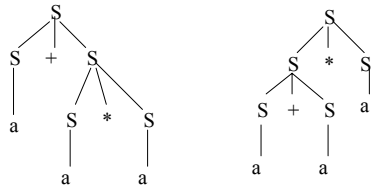
- Another derivation for a + a * a
  - $S \Rightarrow S * S$ rule 3
    $S \Rightarrow S * a$ rule 6
    $S \Rightarrow S + S * a$ rule 1
    $S \Rightarrow a + S * a$ rule 6
    $S \Rightarrow a + a * a$ rule 6

Parse Tree



- $S \Rightarrow S * S$ rule 3
  $S \Rightarrow S * a$ rule 6
  $S \Rightarrow S + S * a$ rule 1
  $S \Rightarrow a + S * a$ rule 6
  $S \Rightarrow a + a * a$ rule 6

Parse trees



Same string, 2 derivations

Ambiguity

- A CFG is said to be ambiguous if there is at least 1 string in L(G) having two or more distinct derivations.

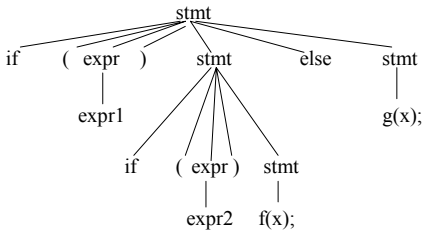
Famous programming language ambiguity

- Dangling else
  - <stmt> → if (<expr>) <stmt> |
    if (<expr>) <stmt> else <stmt> |
    <some_other_stmt>

```
if (expr1) if (expr2) f(); else g();
```

To which `if` does the `else` belong?

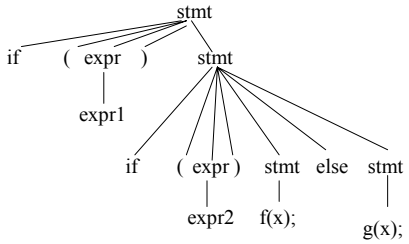
Famous programming language ambiguity



In this derivation, the `else` belongs to the 1st `if`

```
if (expr1) if (expr2) f(); else g();
```

Famous programming language ambiguity



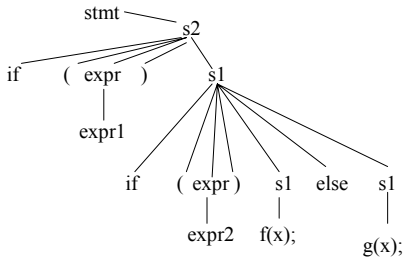

```
if (expr1) if (expr2) f(); else g();
```

Famous programming language ambiguity

- A way to fix this
  - <stmt> → <s1> | <s2>
    <s1> → if (<expr>) <s1> else <s1> | <otherstmt>
    <s2> → if (<expr>) <stmt> |
    if (<expr>) <s1> else <s2>

<s1> represents if statements with matching else
<s2> represent if statements with at least 1 unmatched if
The <s1> in the rule for <s2> will assure that all statements between if and else will not have a dangling else.

Famous programming language ambiguity




```
if (expr1) if (expr2) f(); else g();
```

Ambiguity

- Some languages are inherently ambiguous
  - All possible grammars that generate the language are ambiguous
- Unfortunately, there is no algorithm that can tells us whether a grammar is ambiguous or not.

Ambiguity

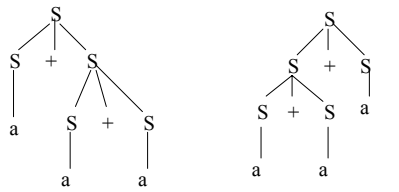
- Showing a grammar is ambiguous is easy
  - Find a string x in the L(G) that has two derivations
- Showing a particular grammar is <u>not</u> ambiguous is usually difficult.
- Showing that any grammar is not ambiguous is not possible.

Derivations

- Leftmost derivations
  - A <u>leftmost derivation</u> is one where the leftmost variable in the current string is always the first to get replaced via a production rule.
  - A <u>rightmost derivation</u> is one where the rightmost variable in the current string is always the first to get replaced via a production rule.

Derivations

a + a + a



rightmost derivation

leftmost derivation

Ambiguity

- As it turns out (we won't prove this)
  - In unambiguous grammars,leftmost derivations will always be unique.
  - In unambiguous grammars,rightmost derivations will always be unique.

Removing ambiguities

- Since some languages are inherently ambiguous
  - This cannot always be done
- In fact,
  - We can/will show there is no "algorithm" for determining if a CFG is ambiguous
- However,
  - On a case by case basis, ambiguities can be eliminated

Example

- Abbreviated grammar for algebraic expressions – Production rules
  - $S \rightarrow S + S$ (1)
    $S \rightarrow S * S$ (2)
    $S \rightarrow (S)$ (3)
    $S \rightarrow a$ (4)

Example

- This grammar has two problems
  1. Precedence of operators is not respected
     a * a + a should be interpreted as (a*a) + a
  2. Sequence of identical operators can be grouped either from the left or the right
     a + a + a can be interpreted as either (a+a)+a or a + (a + a)

Example

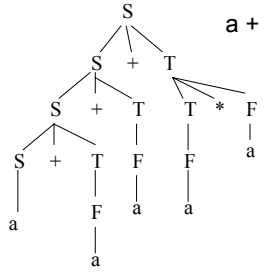
- Solution
  - Introduce some new variables
    - Factor – expression that cannot be broken up by either * or +
      - a
      - (S)
    - Term – expression that cannot be broken up by +
      - All Factors
      - T * F
    - Expression – all possible expression
      - All Terms
      - S + T

Example

- Our new grammar
  - $S \rightarrow S + T \mid T$
  - $T \rightarrow T * F \mid F$
  - $F \rightarrow (S) \mid a$
- Note that
  - all recursion is leftmost
  - * has higher precedent than +
  - a + a + a + a * a is interpreted as
    - ((a+a) + a) + (a*a)

Example

a + a + a + a * a



Example

- It can be shown
  - That every string x, that is generated by this new grammar, has only one leftmost derivation
  - As such this new grammar is unambiguous
  - Done using induction on the |x|.

Summary

- Ambiguity
  - A grammar is ambiguous if there is a string generated by the grammar that has two distinct derivations.
  - Some languages are inherently ambiguous
    - All grammars that generate the language are ambiguous
  - There is no algorithm to determine if any given grammar is ambiguous
    - Proving a grammar to be ambiguous is easy
    - Proving that a grammar is not is hard.
  - Questions?

Summary – Today

- Context Free Grammars
- Parse Trees and Ambiguity

Next time

- Exam 1
- CFG Problem Session