Languages
Math for CS

Grammar

- A phrase structure grammar,

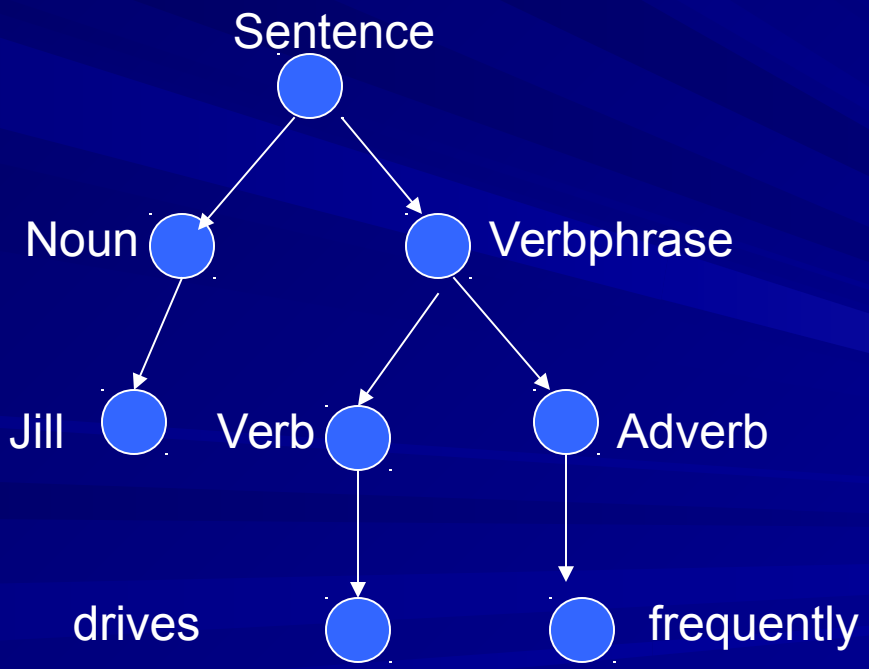
  $G=(V,S, v_0,\rightarrow)$

  - S : terminal symbols
  - N: nonterminal symbols
  - V= S U N
  - $v_0 \in N$
  - $\rightarrow$ a relation on V*

Example

- S={ John, Jill, drives, jogs, carelessly, rapidly, frequently}
- N={sentence, noun, verbphrase, verb, adverb}
- V_0=sentence
- Define →
  - sentence → noun verbphrase
  - noun → John
  - noun → Jill
  - verbphrase → verb adverb
  - verb → drives
  - verb → jogs
  - adverb → carlessly
  - adverb → rapidly
  - adverb → frequently
- Jills drives frequenly (allowable)

Derivation Tree for “Jill drives Frequently”



Parsing

- Parsing is the opposite process of derivation.
  - Given a sentence, is it valid?
  - Take the sentence and constructs a derivation tree that will produce it
  - Is “John drives rapidly” a valid sentence?
- Bottom up parser, top down parser
- Two popular LR parser, and LL parsar
  - LL based grammars : Europe
    - Top down
    - http://en.wikipedia.org/wiki/LL_parser
    - It parses the input from **L**eft to right, and constructs a **L eftmost derivation** of the sentence
    - Expands nonterminals
  - LR based grammars : Everywhere else
    - Bottom up
    - http://en.wikipedia.org/wiki/LR_parser
    - LR parsers read their input from **L**eft to right and produce a **R ightmost derivation**
    - Reduces nonterminals

LL (Top Down)

(1) $S \rightarrow F$

(2) $S \rightarrow (S + F)$

(3) $F \rightarrow 1$

- 1+1
  - $S \rightarrow (S + F) \rightarrow (F + F) \rightarrow (1 + F) \rightarrow (1 + 1)$

Program Parser

- VB, C++ etc parser
- Transforms input text into a tree which shows the structure of the text.
- The tree is then searched and at each step corresponding sentences are generated into another language (machine language code)
- This process is called compiling and is done by a compiler.
- Compiler takes the trees created by the parser and generates machine code.

Types of grammars

- Type 0 no restrictions
- Type 1
  - In each production, a → b, the length of a <= length of b
- Type 2 Context-free grammars
  - Left hand side of each production is a single, nonterminal symbol and the right hand side consists of one or more symbols
- Type 3 Regular grammars
  - Left hand side of each production is a single, nonterminating symbol,
  - Right hand side has one or more symbols
  - Right hand side has at most one nonterminating symbol, which must be to the extreme right of the string
- A language is called Type 3 if there is a grammar of type 3 which produces it.

Grammars

- Post programming languages are of at least Type 2.
- Fast and efficient parsers can be created for Type 2 and Type 3
  - Why, one reason you can parse Type 2 and Type 3 sentences left to right (type 0 and type 1 do not guarantee this)
- The grammar can not remember a construct in an arbitrarily long sentence.
  - This is why one must declare variables in Visual Basic or C++ before one uses the variable.

Regular Grammars

- Theorem. Let S be a finite set, $L \subseteq S^*$. Then L is a regular set if and only if L=L(G) for some regular grammar $G=(V,S, v_0,\rightarrow)$.
- Remember a regular set is a set which can be described by a regular expression.
- Last time we saw a connection between FSM and regular expressions.
- http://en.wikipedia.org/wiki/Regular_language