Lecture 6: Context-free Grammars

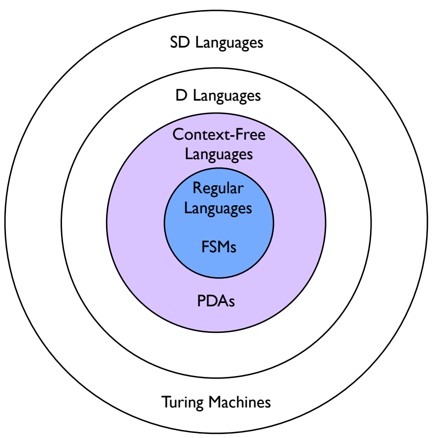
CSCI 81
Spring, 2013

Jason Waterman

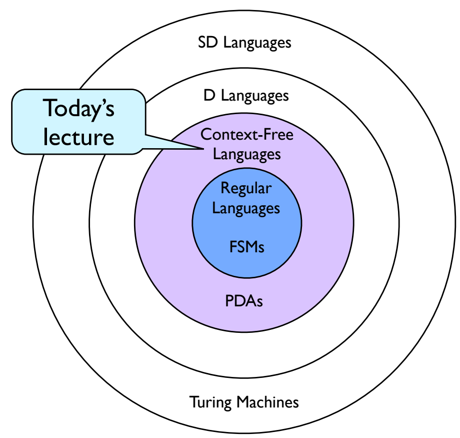
Today: Context-free Grammars

- CFLs in the hierarchy of languages
- What is a grammar?
- Formal definition CFGs
- Examples of CFLs
- Closure properties of CLFs
- Parse trees and how they relate to derivations
- Ambiguous grammars

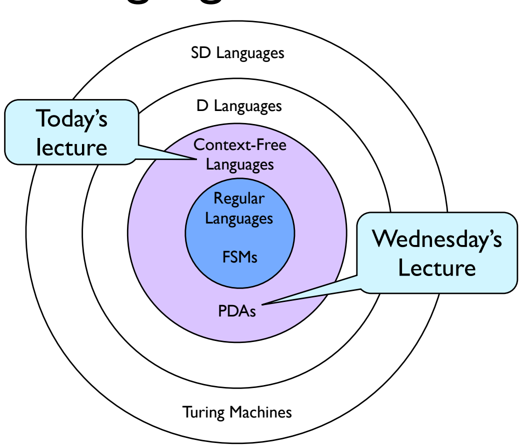
Language Classes



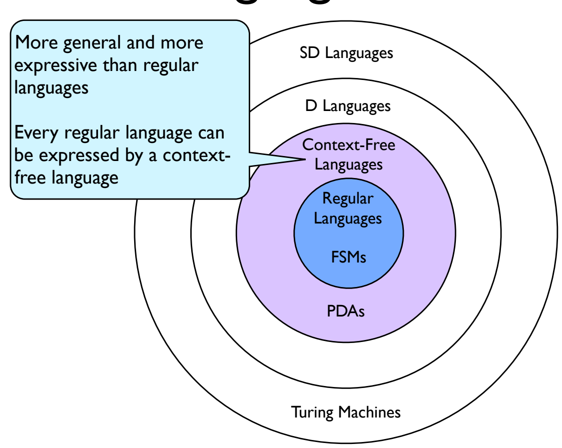
Language Classes



Language Classes



Language Classes



CFGs Applications

- CFGs were first used to study human languages
  - A large portion of the English language can be described by a CFG
- For most programming languages, the set of syntactically legal statements is a CFL
  - Markup languages like HTML and XML
- Used as the basis for compiler design and implementation

What is a Grammar?

- Rewrite system: a list of rules and an algorithm for applying them
- *simple-rewrite*(*R*:rewrite system, *w*: initial string) =
  1. Set *working-string* to *w*
  2. Until told by *R* to halt do:
     - 2.1. Match the left-hand side of some rule against some part of of *working-string*.
     - 2.2. Replace the matched part of *working-string* with the right-hand side of the rule that was matched
  3. Return *working-string*
- If *simple-rewrite(R,w)* can return some string *s* then *R* can derive *s* from *w*
- A rewrite system used to define a language is called a *grammar*

What is a Grammar?

- Rewrite system: a list of rules and an algorithm for applying them
- *simple-rewrite*(*R*:rewrite system, *w*: initial string) =
  1. Set *working-string* to *w*
  2. Until told by *R* to halt do:
     - 2.1. Match the left-hand side of some rule against some part of of *working-string*.
     - 2.2. Replace the matched part of *working-string* with the right-hand side of the rule that was matched
  3. Return *working-string*
- If *simple-rewrite(R,w)* can return some string *s* then *R* can derive *s* from *w*
- A rewrite system used to define a language is called a *grammar*

How do we decide when to halt?

What is a Grammar?

- Rewrite system: a list of rules and an algorithm for applying them
- *simple-rewrite*(*R*:rewrite system, *w*: initial string) =
  1. Set *working-string* to *w*
  2. Until told by *R* to halt do:
     - 2.1. Match the left-hand side of some rule against some part of of *working-string*.
     - 2.2. Replace the matched part of *working-string* with the right-hand side of the rule that was matched
  3. Return *working-string*
- If *simple-rewrite(R,w)* can return some string *s* then *R* can derive *s* from *w*
- A rewrite system used to define a language is called a *grammar*

How do we decide when to halt?

When matching multiple rules which one do we pick?

Context-free Grammars

- Formally, a context-free grammar G is a quadruple(V, Σ, R, S), where:
  - V is the rule alphabet (containing both terminals and non-terminals)
  - Σ (the set of terminals) is a subset of V
  - R (the set of rules) is finite subset of (V - Σ) × V*
  - S (the start symbol) can be any element of V - Σ

Derivability relation

- Given a grammar *G*, define $x \Rightarrow y$ to be the binary relation *derives-in-one-step* such that:
  - $\forall x,y \in V^*(x \Rightarrow y$ iff $x = \alpha A \beta$, $y = \alpha \gamma \beta$, and there exists a rule $A \rightarrow \gamma$ in $R_G)$
- Any sequence of the form: $w_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n$ is called a *derivation*
  - $\Rightarrow^*$ is the reflexive, transitive closure of $\Rightarrow$ and is called the *derives* relation
- The language *generated* by G, denoted L(G), is the set of all string terminals that an be derived from a starting symbol S using zero or more applications of rules in G

Balanced Parenthesis Language

- Bal = {$w \in$ {), (, }* : the parenthesis are balanced}
- Not regular but is context free
- G = ({S,), (}, R, S), where:
  - R = {S → (S), S → SS, S → ε}

Example CFG: A^nB^n

- $A^nB^n = \{a^nb^n : n \geq 0\}$
- Not regular
- G = {{S, a, b}, {a, b}, R, S}
  - What does this grammar generate?
  - What is R?

Example CFG: A^nB^n

- $A^nB^n = \{a^nb^n : n \geq 0\}$
- Not regular
- G = {{S, a, b}, {a, b}, R, S}
  - What does this grammar generate?
  - What is R?
  - R = {S → aSb, S → ε}

Regular vs. Context-free Grammars

- In a regular grammar, every rule must have:
  - A left hand side that is a single nonterminal
  - A right hand side that is ε or a single terminal or a single terminal followed by a single nonterminal
- In a context-free grammar, every rule must have:
  - A left hand side that is a single nonterminal
  - A right hand side

Regular vs. Context-free Grammars

- In a regular grammar, every rule must have:
  - A left hand side that is a single nonterminal
  - A right hand side that is ε or a single terminal or a single terminal followed by a single nonterminal
- In a context-free grammar, every rule must have:
  - A left hand side that is a single nonterminal
  - A right hand side

These extra conditions make regular grammars less expressive and a subset of CFGs

Closure Properties of CFGs

- Union
  - Suppose we have grammars for two languages, with start symbols S and T
  - Rename variables as needed to ensure that the two grammars don't share any variables
  - Then construct a grammar for the union of the languages, with start symbol Z, by taking all the rules from both grammars and adding a new rule $Z \rightarrow S \mid T$
- Concatenation
  - Similar to union: rename variables as needed, take all rules from both grammars, and add new rule Z → ST

Closure Properties of CFGs

- Kleene Star
  - Suppose we have a grammar for the language L, with start symbol S. The grammar for L*, with start symbol T, contains all the rules from the original grammar plus the rule $T \rightarrow TS \mid \varepsilon$
- String Reversal
  - Reverse the character string on the righthand side of every rule in the grammar

Closure Properties of CFGs: Substitution

- If a substitution *s* assigns a CFL to every symbol in the alphabet of CFL *L*, then *s(L)* is a CFL
  - Take a grammar for *L* and a grammar for each language $L_a = s(a)$
  - Make sure all the variables of all these grammars are different
  - Replace each terminal a in the production rules for *L* by S_a, the start symbol of the grammar for L_a
  - This replacement allows any string in L_a to take the place of any occurence of *a* in any string of *L*

Derivations and Parse Trees

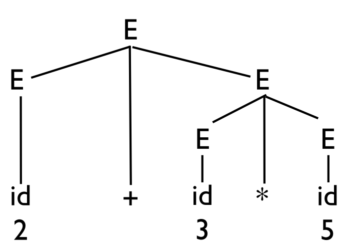
- CFGs do more than just describe the set of strings in a language
- They provide a way of assigning an internal structure (via their derivations) to the strings they describe
- This allows us to assign *meanings* to the strings a grammar can produce
- This grammatical structure of a string is captured by a *parse tree*
  - A record of which rules were applied to which nonterminals during the string's derivation

Parse Tree Definition

- A parse tree derived by a grammar $G=(V, \Sigma, R, S)$ is a rooted ordered tree
  - Every leaf node is labeled with an element of $\Sigma \cup \{\varepsilon\}$
  - The root node is labeled S
  - Every other node is labeled with some element of $V - \Sigma$ (that is a nonterminal symbol)
  - If *m* is a nonleaf node labeled *X* and the children of *m* are labeled $x_1, x_2, \ldots x_n$ then *R* contains the rule $X \rightarrow x_1, x_2, \ldots x_n$

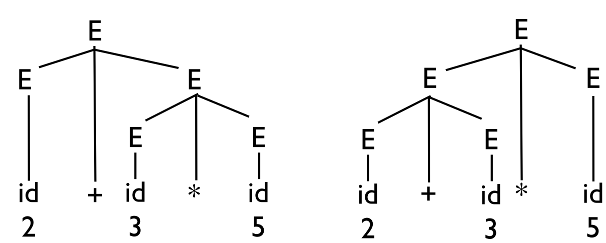
Parse Tree for a Simple Expression Grammar

- Define E_{xpr} with the following CFG
  - $G = (\{E, \text{id}, +, *\}, \{\text{id}, +, *\}, R, E)$
  - $R = \{E \rightarrow E + E, E \rightarrow E * E, E \rightarrow \text{id}\}$
- Parse tree for 2 + 3 * 5:



Ambiguity

- For E_{xpr} there is more than one parse tree for the string 2 + 3 * 5
- Why is this a problem?



Inherent Ambiguity

- If there exists a CLF for which no unambiguous grammar exists, we call such languages *inherently ambiguous*
  - It is possible to construct a new grammar *G'* that generates *L(G)* that has less (or no) ambiguity
- Unfortunately, given a CFG *G*, determining if *G* is ambiguous is undecidable

Unambiguous Expression Grammar

- *G'* = ({*E*, T, F, id, +, *}, {id, +, *}, *R*, *E*)
- R = {E → E + T, E → T, T → T * F, T → F, F → id}
  - By adding the levels T (for term) and F (for factor) we have defined a precedence hierarchy
  - Now there is a single parse tree for id + id * id