Lecture 4 (context free languages)

Thursday, February 23, 2023 11:16 AM

- Context free langs
  - Using grammar rules to define a lang
  - Context free grammars: Definitions and more examples
  - Regular langs and regular grammars
  - Derivation trees and ambiguity
  - Simplified forms and normal forms
- Using grmamer rules to define a lang
  - Reg languages and FA too simple
    - Using context free grammars lets us describe more interesting langs
    - Much high lvl pgrming lang syntax can be expressed with context-free grammars
    - Context-free grammars with very simple form given another way to describe the reg lang
    - Terminal symbols and stuff
    - Recursive rule
    - Termination and recursive state
  - Grammars can be ambiguous
  - Usually use a(^n)b(^n), pumping lemma shows not regular lang
  - Grammer is set of rules, usually simpler than those of English, by which strings in a lang can be generated
  - Recursive case, then termination case
  - Consider the language $AnBn = \{a^n b^n \mid n \geq 0\}$, defined using the recursive definition:
    - $\Lambda \in AnBn$
    - For every $S \in AnBn$, $aSb \in AnBn$
  -
  -
  - If alpha and beta are strings, alpha...alpha => beta means beta is got from a in one step, from rule replace single occurrence of S by either lambda or aSb (rule can be states as S->lambda|aSb)
  - S sometimes called erasing rules
  - | means such that, or 'or'
    - Means option
    - Means such that
  - Lft side has same non-terminal var
    - Lft side is called nonterminal
    - Rght side is terminal
- Def and more Ex
  - Def
  - **Definition:** A *context-free grammar* (CFG) is a 4-tuple $G=(V, \Sigma, S, P)$, where V and Σ are disjoint finite sets, $S \in V$, and P is a finite set of formulas of the form $A \to \alpha$, where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$. ■
    - Elements of Σ are *terminal symbols*, or *terminals*, and elements of V are *variables*, or *nonterminals.*
    - S is the *start* variable, and elements of P are *grammar rules*, or *productions.*
    - We use $\to$ for productions in a grammar and $\Rightarrow$ for a step in a derivation.
    - The notations $\alpha \Rightarrow^n \beta$ and $\alpha \Rightarrow^* \beta$ refer to n steps and zero or more steps, respectively.
  -
  - $\Rightarrow_G$
  - This indicates a derivation in a certain grammar G
  - Context free = left side (A) can be applied wherever A occurs in the string (no matter context)
  - Automaton was 5 tuple, grammar is 4 tuple
  - Elements of sigma are terminal symbols (terminals or letters), elements of V are variables (nonterminal)
  - S is start var, elements of P are grammar rules, or productions
  - Use -> for productions in a grammar, => means step in a derivation
    - Many arrows means thick derivation, can grow to word with 1 var, stop when case reached
  - Def of result/derive/single step in a derivation
    - A=>B means strings a1, a2, and gamma in (V U sigma)*
  - Context free means...it don't need context
  - Lang generated def
  -

Definition: If $G = (V, \Sigma, S, P)$ is a CFG, the language generated by G is

$$L(G) = \{ x \in \Sigma^* \mid S \Rightarrow_G^* x \}$$

- (S is the *start* variable, and x is a string of terminals). ■

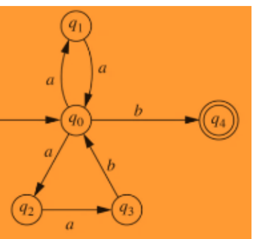
  A language L is a *context-free language* (CFL) if there is a CFG G with $L = L(G)$. ■
- AEqB = {x (is element of) {a,b}*|n_a(x) = n_b(x)}
  - Make CFG for AEqB
  -
-
- What is the meaning of context free?
  - **Answer:** What makes the grammar *context-free* is that the production above, with left side A, can be applied wherever A occurs in the string (irrespective of the context; i.e., regardless of what α_1 and α_2 are).
-
- **Definition:** If $G = (V, \Sigma, S, P)$ is a CFG, the language generated by G is

  $L(G) = \{ x \in \Sigma^* \mid S \Rightarrow_G^* x \}$

  (S is the *start* variable, and x is a string of terminals). ■

  A language L is a *context-free language* (CFL) if there is a CFG G with $L = L(G)$. ■
-
- Ex: AEqB(literally A equals B)= look at pic
  - S -> lambda | aB | bA
  - A -> a|bAA
  - B -> b|aBB
  - Consider $AEqB = \{x \in \{a,b\}^* \mid n_a(x) = n_b(x)\}$

    Let us design a CFG for $AEqB$.
  -
  - If x is a non-null string in $AEqB$ then either $x = ay$, where $y \in L_b = \{z \mid n_b(z) = n_a(z) + 1\}$, or $x = by$, where $y \in L_a = \{z \mid n_a(z) = n_b(z) + 1\}$.
- Theorem:
- If L1 and L2 (2 languages) are CFL's (context free langs) over sigma, then so are L1 U L2, L1L2, and L1*
  - All the U, Concatenation, and * are closed under these operations under these languages
- Reg Langs and Reg Grammars
  - Prove by structural induction that every reg lang over sigma is a CFL
  - **Definition 4.13:** A context-free grammar is *regular* if every production is of the form $A \to \sigma B$ or $A \to \Lambda$. ■
  -
  - What is the regular grammar corresponding to the FA on the right? (get a *)
    - Let S, A, B, C, and D be the non-terminals corresponding to states q_0, q_1, q_2, q_3, and q_4.
    - S → *a*A | *a*B | *b*D
    - A → *a*S
    - B → *a*C
    - C → *b*S
    - D → Λ ■

  -
  - Slide 17
- Deviation Trees and Ambiguity
  -
  - **Definition 4.18:** A CFG G is *ambiguous* if, for at least one $x \in L(G)$, x has more than one derivation tree (or equivalently, according to Theorem 4.17, more than one LMD). ■
  -
  - Root node reps start var
  - Interior node and its kids rep a production A -> alpha used in the derivation, node reps A and kids (left to right), rep symbols in alpha
  -

  -
  - Slide 23
  - It goes to second if statement
  - 2 derivation trees, 2 interpretations of dangling else
  - Grammar is ambiguous, but can let correct interpretation w/ equivalent grammars
  - S goes to S2, goes to if(E)S, goes to if(E)S1, goes to
  - Consider the CFG G: S-> S+S|S*S|(S)|a
  - redesign bc PEMDAS
  -
- Simplified Forms and Normal Forms
  - unambiguous CFG making L(G)
  - ex: grammar w/ no sigma productions and no unit productions, can deduce no derivation of astring x can take more than 2|x| - 1
  -
  - //Simplified & Normal Forms
  -
  - productions A->BCDC, B->lamda, and C-> lambda
  - B and C are nullable variable
  -
  - Suppose we have the productions $A \to BCDC$, $B \to \Lambda$, and $C \to \Lambda$.
    - If we get rid of Λ-productions, then the steps that replace B and C by Λ will no longer be possible, but we must still be able to get all the same non-null strings from A.

    We must retain the production $A \to BCDC$ but we should add instead:
    - $A \to CDC \mid BDC \mid BCD \mid DC \mid CD \mid BD \mid D$.
  -
  - Definition__
  -
  - **Definition 4.26:** A recursive definition of the set of nullable variables of G.
    - If there is a production $A \to \Lambda$ then A is nullable.
    - If $A_1, A_2, \ldots, A_k$ are nullable variables and there is a production $B \to A_1A_2\ldots A_k$, then B is nullable. ■
  -
  - leads immediately to algorithm for id the null vars
- Slide 35
- Ex:
  - Consider G', 2 unit productions, I think | means or
  - Let G = ({S}, {a, b}, S, {S → aSb | Λ}).

    (get a star) What is the equivalent CFG without Λ-productions?
  - The last lambda just becomes |ab
  -
  - (get a star) How about G = ({S, A}, {a, b, c}, S, {S → cS | aAb, A → aAb | Λ})?
  - Again, just |ab
  - (get a star) What is the language generated by the second CFG?

    L(G) = $\{c^m a^n b^n \mid m \geq 0, n \geq 1\}$. ■
  -
  - Let us consider G' = ({S, A, C}, {a, b, c}, S, {S → C, C → cC | A, A → aAb | ab}).

    We have two unit productions: S → C, and C → A.

    We get two trivial derivations: S ⇒ C, C ⇒ A and by transitivity, we get a new S-*derivable* one: S ⇒* A.
  -
  - The equivalent CFG is given by:
    - G' = ({S, A}, {a, b, c}, S, {S → cC | aAb | ab, C → cC | aAb | ab, A → aAb | ab}). ■
  -
- Simplified and normal forms
  - Chomsky normal form, if every productions is of one of these two types:
    - A-> BC (B and C are vars
    - A-> sigma (where delta is a terminal)
  - For every context-free grammar G, there is another CFG G1 in Chomsky normal form such that L(G1) = L(G) - {A}
  - First step is elim lambda production and unit productions
  - second step is intro every terminal symbol sigma a new var Xsigma and production Xsigma -> sigma

  - Every production, replace every terminal by new var (except for new productions above)
  - Replace a production like A -> BACB by the productions A->BY1, Y1->AY2, Y2->CB, where Y1 and Y2 are new vars
  - CFG in Chomsky normal form is CNF
- Ex of getting a CNF
  - Good luck
  - G=({S,X}, {a, b}, S, {S → ab | aXb, X → ab | aXb})
  - Replace each terminal symbol by non-terminal symbol and a rule that goes into the terminal symbol:
    - S → AB | AXB, X → AB | AXB, A → a, B → b
    - The rules S → AB, X → AB, A → a, B → b are already in Chomsky Normal Form.

    We need to transform the other rules, these are, S → AXB, X → AXB.

    The rule S → AXB is transformed into S → AY, Y → XB.

    The rule X → AXB is transformed into X → AZ, Z → XB.

    Note: Since XB repeats twice, we can have instead X → AY.
  - The final CFG: S → AB | AY, Y → XB, X → AB | AY, A → a, B → b. ■
- Summary
  - Context-Free Languages
    1. Using Grammar Rules to Define a Language
    2. Context-Free Grammars: Definitions and More Examples
    3. Regular Languages and Regular Grammars
    4. Derivation Trees and Ambiguity
    5. Simplified Forms and Normal Forms