COMS3261:
Computer Science Theory

Fall 2013

Mihalis Yannakakis

Lecture 14, 10/23/13

CLOSURE PROPERTIES OF CFL's

- CFLs closed under Concatenation:

  L_1, L_2 CFL $\Rightarrow$ $L_1 \cdot L_2$ CFL

  Proof: Given CFGs G_1,G_2 for L_1, L_2 .

  Wlog assume G_1, G_2 have disjoint sets of variables V_1,V_2 and start symbols S_1, S_2

  New grammar G:

  Variable set $V = V_1 \cup V_2$ + new start variable S

  Productions = union of the productions and $S \rightarrow S_1 S_2$

Examples: Concatenation

1. $L=\{ a^n b^n c^m \mid n, m \geq 0\}$ is a CFL:
   concatenation of $L_1 = \{a^n b^n \mid n \geq 0\}$ and $L_2 = \{c^m \mid m \geq 0\} = c^*$
   Grammar: $S \rightarrow S_1 S_2$
   $S_1 \rightarrow \varepsilon \mid aS_1 b$
   $S_2 \rightarrow \varepsilon \mid cS_2$

2. $L=\{ a^i b^j a^k \mid j=i+k;\ i,j,k \geq 0\}$ is CFL:
   concatenation of $L_1 = \{ a^i b^i \mid i \geq 0\}$ and $L_2 = \{ b^k a^k \mid k \geq 0\}$
   Grammar: $S \rightarrow S_1 S_2$
   $S_1 \rightarrow \varepsilon \mid aS_1 b$
   $S_2 \rightarrow \varepsilon \mid bS_2 a$

Closure under Union

- CFLs closed under Union: L_1, L_2 CFL $\Rightarrow$ $L_1 \cup L_2$ CFL
  Proof: Similar to concatenation
  Given CFGs G_1, G_2 for L_1, L_2 .
  Wlog assume G_1, G_2 have disjoint sets of variables V_1, V_2 and start symbols S_1, S_2

  New grammar G:
  Variable set $V = V_1 \cup V_2$ + new start variable S
  Productions = union of the productions and $S \rightarrow S_1 \mid S_2$

Example : Union

- Example: $L = \{ a^i b^j c^k \mid i=j \text{ or } j=k \text{ ; } i,j,k \geq 0 \}$ is CFL
  $L = \{ a^i b^j c^k \mid i = j \} \cup \{ a^i b^j c^k \mid j=k \}$
- Grammar for $L_1 = \{ a^i b^j c^k \mid i = j \}$ (a shorter version)
  $S_1 \rightarrow T \mid S_1c$
  $T \rightarrow \varepsilon \mid aTb$ [T derives all strings of form a^ib^i]
- Grammar for $L_2 = \{ a^i b^j c^k \mid j = k \}$
  $S_2 \rightarrow R \mid aS_2$
  $R \rightarrow \varepsilon \mid bRc$ [R derives all strings of form b^kc^k]

- Grammar for L
  $S \rightarrow S_1 \mid S_2$
  $S_1 \rightarrow T \mid S_1c$
  $T \rightarrow \varepsilon \mid aTb$
  $S_2 \rightarrow R \mid aS_2$
  $R \rightarrow \varepsilon \mid bRc$

Closure Under Star

- CFLs closed under Star *: L_1 CFL $\Rightarrow$ $(L_1)^*$ CFL
  Proof: Take grammar G_1 for L_1 with start symbol S_1
  Add new start symbol S and production $S \rightarrow \varepsilon \mid S_1S$

  Example: The set of strings of the form
  $a^{n1}b^{n1}\ a^{n2}b^{n2}\ a^{n3}b^{n3} \ldots\ a^{nk}b^{nk}$ is CFL
  $= \{ a^nb^n \mid n \geq 0\}^*$

  Grammar: $S \rightarrow \varepsilon \mid S_1S$
  $S_1 \rightarrow \varepsilon \mid aS_1b$

Closure under Homomorphism

- Homomorphism: Mapping h from an alphabet Σ to set of strings over an alphabet T (same or different)
- $\forall a \in \Sigma$, $h(a) \in T^*$
- Extend map to strings: $h(a_1 \ldots a_n) = h(a_1) \ldots h(a_n) \in T^*$
- Extend to languages L over Σ : $h(L) = \{ h(w) \mid w \in L \}$

Generalization: Closure under Substitution

- Substitution: Mapping s from an alphabet Σ to set of languages over an alphabet T (same or different)
- $\forall a \in \Sigma$, $s(a)=L_a$, a language over T
- Extend map to strings: $s(a_1 \ldots a_n) = s(a_1) \ldots s(a_n) = L_{a_1} \ldots L_{a_n}$
  - Note: s(string) is a language, not a string
- Extend to languages L over Σ : $s(L) = \bigcup_{w \in L} s(w)$

- Example: $\Sigma = \{1,2\}$, $T=\{a,b,c\}$, $s(1) = \{a^n b^n | n \geq 1\}$, $s(2)=\{bc\}$
  $s(12) = \{a^n b^{n+1} c | n \geq 1\}$,
  $s(11) = \{a^{n1} b^{n1} a^{n2} b^{n2} \mid n_1, n_2 \geq 1 \}$
  $s(1^*) = \{a^{n1} b^{n1} a^{n2} b^{n2} \ldots a^{nk} b^{nk} \mid k \geq 0, n_i \geq 1 \}$

Substitution examples

Union, Concatenation, Star are special cases of Substitution applied to languages

$\Sigma = \{1,2\}$, $s(1) = L_1$, $s(2) = L_2$

Union: $s(\{1,2\}) = L_1 \cup L_2$

Concatenation: $s(\{12\}) = L_1 L_2$

Star: $s(\{1\}^*) = (L_1)^*$

CFL Closure under Substitution

- Theorem: If L is a cfl over alphabet Σ, and $s(a)=L_a$ is cfl for all $a \in \Sigma$, then $s(L)$ is cfl.
- Proof: Take grammars $G=(V,\Sigma,P,S)$ for L, and grammars (V_a,T,P_a,S_a) for each L_a. Assume wlog all variable sets disjoint by renaming them if necessary.
- Replace each occurrence of each terminal a in productions of G by S_a.
- Resulting cfg $G'=(V',T,P',S)$ where $V'= V \cup \bigcup_{a \in \Sigma} V_a$,

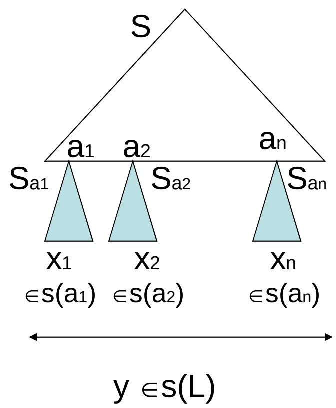
  $P'=$ modified $P \cup \bigcup_a P_a$
- Example: 1^* generated by $S \rightarrow \varepsilon \mid 1S$

  $s(1) = L_1 = \{a^n b^n \mid n \geq 0\}$ generated by $S_1 \rightarrow \varepsilon \mid aS_1b$

  CFG for $(L_1)^*$: $S \rightarrow \varepsilon \mid S_1 S$, $S_1 \rightarrow \varepsilon \mid aS_1b$

Proof of Theorem

- Parse trees of G': Take a parse tree T of G, replace each leaf labeled $a \in \Sigma$ by a parse tree T_a in G_a (several leaves of T may have same label a, but the trees can be different)



$y \in s(L) \Leftrightarrow y \in L(G')$

Both ways:

$y \in s(L) \Rightarrow \exists a_1 \ldots a_n \in L$,
$\exists x_1 \in L_{a1}$, $x_n \in L_{an}$ s.t. $y = x_1 \ldots x_n \Rightarrow$
can construct a parse tree like that

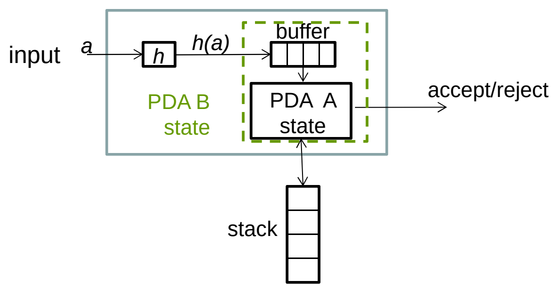
$y \in L(G') \Rightarrow$ its parse tree can be decomposed like that

Closure under Reversal

- Can't show with substitution theorem, but can show directly
- Given grammar G=(V,T,P,S), define $G^R = (V,T,P^R,S)$, where P^R reverses the bodies of all the productions of P, i.e. $A \to \alpha$ becomes $A \to \alpha^R$
- Then every derivation of G, $S \Rightarrow \ldots. \Rightarrow w$ yields a corresponding derivation in G^R where all sentential forms are reversed $S \Rightarrow \ldots. \Rightarrow w^R$

Closure under Inverse Homomorphism

- If $h:\Sigma \rightarrow T^*$ is a homomorphism and L a language, then $h^{-1}(L) = \{ w \in \Sigma^* \mid h(w) \in L \}$
- Theorem: If L is a CFL then $h^{-1}(L)$ is also a CFL
- Proof:
- Use a PDA A for L to construct a PDA B for $h^{-1}(L)$



NON-CLOSURE Properties of CFLs

- Not closed under: Intersection, Complement, Difference

- Intersection:
- $L_1 = \{ a^n b^n c^i \mid n \geq 0, i \geq 0\}$ is CFL,
  $L_2 = \{ a^i b^n c^n \mid n \geq 0, i \geq 0\}$ is CFL, but
  $L_1 \cap L_2 = \{ a^n b^n c^n \mid n \geq 0 \}$ is not CFL

CFL Nonclosure ctd.

- Complement:

  Proof 1: By DeMorgan's law $L_1 \cap L_2 = (L_1^c \cup L_2^c)^c$

  If CFL were closed under complement, and since they are closed under $\cup$, then they would be also closed under $\cap$

  Proof 2: $\{ a^n b^n c^n \mid n \geq 0 \}$ is not CFL

  complement = $(a^*b^*c^*)^c \cup \{a^i b^j c^k \mid i \neq j\} \cup \{a^i b^j c^k \mid j \neq k\}$ is CFL

- Difference: $L_1 - L_2$

  Proof: Take $L_1 = \Sigma^*$. Then $(L_2)^c = L_1 - L_2$

- DCFLs are closed under complement

  Switch final, nonfinal states of DPDA & eliminate first nonterminating computations (nontrivial)

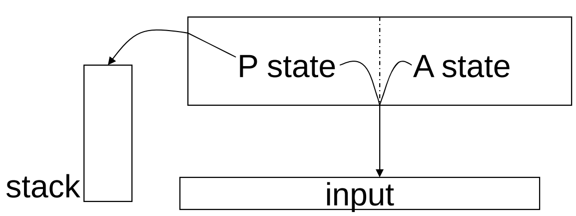
CFL CLOSURE under $\cap$ with Regular

- If L is CFL and R is a regular language then $L \cap R$ is CFL Also L-R is CFL (caution: but R-L is not necessarily CFL)
- Proof: Take PDA P for L, FA A for R.
- Product construction:

  Construct PDA M that follows in parallel the computation of both on a given input w (similar to $\cap$ of regular languages).

  State of M keeps track of both the state of P and R

  $M = P \times A$ has state set $Q_M = Q_P \times Q_A$, Final set $F_M = F_P \times F_A$



ALGORITHMS for CFLs

- We saw already several algorithms:
- Conversion between PDAs that accept by final state or empty stack: Linear time, size
- Conversion from CFG to PDA: Linear time, size
- Conversion from PDA to CFG: $O(n^3)$, where n is the size of the description of the PDA (states, all transitions)

ALGORITHMS for CFLs

- We will discuss algorithms for:
- Emptiness of a language of a CFG:
  Given CFG G, is $L(G)=\emptyset$?
  - If CFL given by PDA, can convert to CFG and test the CFG
- Cleaning algorithms for CFG
  e.g. elimination of useless variables and productions
- Transformation to a simple form: Chomsky Normal Form
- Membership / Parsing problem
  Given CFG G, string w, does $w \in L(G)$?

Testing Emptiness of CFG

- Given CFG $G=(V,T,P,S)$, is $L(G) = \emptyset$?
- A variable X is called generating if $X \Rightarrow^* w$ for some terminal string w
- $L(G) \neq \emptyset$ iff the start variable S is generating
- We will compute all the generating variables

Algorithm for Computing the Generating Variables

- Initialization (Basis): $K := T$
- Loop (Induction): while ($\exists$ production $X \rightarrow \beta$ such that $X \notin K$ but all symbols of $\beta \in K$) $K := K \cup \{X\}$
- Return the variables in K

- Time: straightforward: $O(|G|^2)$, where $|G|$ is the size of the grammar (includes sum of lengths of the productions)
- Can do in $O(|G|)$ time with more care with appropriate data structure – see book, Sec 7.4.3

Example

- $S \rightarrow ABE \mid AC$
  $A \rightarrow 1B \mid 0C$
  $B \rightarrow 0D$
  $C \rightarrow 1$
  $D \rightarrow AB$
  $E \rightarrow 0$

- K ={0,1}
- Add C, E, A, S
- $\Rightarrow$ B, D not generating

Correctness of Generating Algorithm

- 1. If $X \in$ final set K then X generating
- Proof: by induction on the iteration i that added X to K
  $X \rightarrow \beta$ such that all symbols of $\beta \in K$ in earlier iteration or because they are terminals. By i.h. they can each derive a terminal string $\Rightarrow$ X too.

2. If X generating then $X \in$ final set K

Proof: By induction on length of shortest derivation $X \Rightarrow^* w$

Derivation starts as $X \Rightarrow \beta \Rightarrow^* w$

Every variable of β has a shorter derivation $\Rightarrow$ in K by i.h.
$\Rightarrow$ will add also X to K.