Harvard CS 121 and CSCI E-207

Lecture 12: General Context-Free Recognition

Harry Lewis

October 12, 2010

Reading: Sipser, pp. 119-128.

The Top-Down CFG $\rightarrow$ PDA construction, revisited

Transitions:

- $\delta(q_{\text{start}}, \varepsilon, \varepsilon) = \{(q_{\text{loop}}, S\$)\}$

  "Start by putting $S\$$ on the stack, & go to q_{loop}"

- $\delta(q_{\text{loop}}, \varepsilon, A) = \{(q_{\text{loop}}, w)\}$ for each rule $A \rightarrow w$

  "Remove a variable from the top of the stack and replace it with a corresponding righthand side"

- $\delta(q_{\text{loop}}, \sigma, \sigma) = \{(q_{\text{loop}}, \varepsilon)\}$ for each $\sigma \in \Sigma$

  "Pop a terminal symbol from the stack if it matches the next input symbol"

- $\delta(q_{\text{loop}}, \varepsilon, \$) = \{(q_{\text{accept}}, \varepsilon)\}$.

  "Go to accept state if stack contains only $\$$."

The Dual Bottom-Up CFG $\rightarrow$ PDA Construction

- $\delta(q_{\text{start}}, \varepsilon, \varepsilon) = \{(q_{\text{loop}}, \$)\}$

  "Start by putting \$ on the stack, & go to q_{loop}"

- $\delta(q_{\text{loop}}, \sigma, \varepsilon) = \{(q_{\text{loop}}, \sigma)\}$ for each $\sigma \in \Sigma$

  "Shift input symbols onto the stack"

- $\delta(q_{\text{loop}}, \varepsilon, w^R) = \{(q_{\text{loop}}, A) : A \rightarrow w)$ is a rule of $G\}$

  "Reduce right-hand sides on the stack to corresponding left-hand sides"

- $\delta(q_{\text{loop}}, \varepsilon, S\$) = \{(q_{\text{accept}}, \varepsilon)\}$

  "Accept if the stack consists just of S above the bottom-marker"

Context-Free Recognition

- **Goal:** Given CFG G and string w to determine if $w \in L(G)$
- **First attempt:** Construct a PDA M from G and run M on w.
- **Brute-Force Method:**

  Check all parse trees of height up to some upper limit depending on G and $|w|$

  Exponentially costly
- **Better:**

1. Transform G into Chomsky normal form (CNF) (once for G)
2. Apply a special algorithm for CNF grammars

   (once for each w)

Chomsky Normal Form

Def: A grammar is in Chomsky normal form if

- the only possible rule with ε as the RHS is $S \to \varepsilon$

  (Of course, this rule occurs iff $\varepsilon \in L(G)$)

- Every other rule is of the form

1) $X \to YZ$

   where X, Y, Z are variables

2) $X \to \sigma$

   where X is variable and σ is a single terminal symbol

Transforming a CFG into Chomsky Normal Form

Definitions:

- ε-rule: one of the form $X \to \varepsilon$
- Long Rule: one of the form $X \to \alpha$ where $|\alpha| > 2$.
- Unit Rule : One of the form $X \to Y$

  where $X, Y \in V$
- Terminal-Generating Rule: one of the form $X \to \alpha$

  where $\alpha \notin V^*$ and $|\alpha| > 1$ (α has at least one terminal)

Eliminate non-Chomsky-Normal-Form Rules In Order:

1. All ε-rules, except maybe $S \to \varepsilon$

2. All unit rules

3. All long rules

4. All terminal-generating rules

- While eliminating rules of type j, we make sure not to reintroduce rules of type $i < j$.

Eliminating ε-Rules

0. Ensure start variable does not appear on the RHS of any rule (by adding new start variable if necessary).

1. To eliminate ε-rules, repeatedly do the following:

   a. Pick a ε-rule $Y \to \varepsilon$ and remove it.

   b. Given a rule $X \to \alpha$ where α contains n occurrences of Y, replace it with 2^n rules in which $0, \ldots, n$ occurrences are replaced by ε. (Do not add $X \to \varepsilon$ if previously removed.)

   e.g.

$$X \to aYZbY \qquad \Rightarrow$$

(Why does this terminate?)

Eliminating Unit and Long Rules

2. To eliminate unit rules, repeatedly do the following:

   a. Pick a unit rule $A \to B$ and remove it.

   b. For every rule $B \to u$, add rule $A \to u$ unless this is a unit rule that was previously removed.

3. To eliminate long rules, repeatedly do the following:

   a. Remove a long rule $A \to u_1 u_2 \cdots u_k$, where each $u_i \in V \cup \Sigma$ and $k \geq 3$.

   b. Replace with rules $A \to u_1 A_1, A_1 \to u_2 A_2, \ldots, A_{k-2} \to u_{k-1} u_k$, where $A_1, \ldots,$ A_{k-2} are newly introduced variables used only in these rules.

Eliminating Terminal-Generating Rules

4. To eliminate terminal-generating rules:

   a. For each terminal a introduce a new nonterminal A.

   b. Add the rules $A \to a$

   c. "Capitalize" existing rules, e.g.

      replace $X \to aY$
      with $X \to AY$

Example of Transformation to Chomsky Normal Form

Starting grammar:

$$S \to XX$$
$$X \to aXb|\varepsilon$$

Benefit of CNF for Deciding if $w \in L(G)$

- **Observation:** If $S \Rightarrow XY \Rightarrow^* w$, then $w = uv$, $X \Rightarrow^* u$, $Y \Rightarrow^* v$ where u, v are *strictly shorter* than w.

- **Divide and Conquer:** can decide whether S yields w by recursively determining which variables yield substrings of w.

- **Dynamic Programming:** record answers to all subproblems to avoid repeating work.

Determining $w \in L(G)$, for G in CNF

Let $w = a_1 \dots a_n$, $a_i \in \Sigma$.
Determine sets S_{ij} $(1 \le i \le j \le n)$:

$$S_{ij} = \{X : X \stackrel{*}{\Rightarrow} a_i \dots a_j, X \text{ variable of } G\}$$

	a_1					
S_{11}		a_2				
S_{12}	S_{22}		a_3			
S_{13}	S_{23}	S_{33}				
	S_{24}	S_{34}	S_{44}			
						a_n
S_{1n}					S_{nn}	

- $w \in L(G)$ iff start symbol $\in S_{1n}$

Filling in the Matrix

- Calculate S_{ij} by induction on $j-i$

$$
\begin{array}{ll}
(j-i=0) & S_{ii} = \{X : X \rightarrow a_i \text{ is a rule of } G\} \\
(j-i>0) & X \in S_{ij} \text{ iff } \exists \text{ rule } X \rightarrow YZ \\
 & \qquad\qquad\qquad \exists k : i \leq k < j \\
 & \qquad\quad \text{such that } \; Y \in S_{ik} \\
 & \qquad\qquad\qquad Z \in S_{k+1,j}
\end{array}
$$

e.g. $w = abaabb$

The Chomsky Normal Form Parsing Algorithm

for $i \leftarrow 1$ to n do

$$S_{ii} = \{X : X \rightarrow a_i \text{ is a rule } \}$$

for $d \leftarrow 1$ to $n-1$ do

for $i \leftarrow 1$ to $n-d$ do

$$S_{i,i+d} \leftarrow \bigcup_{j=i}^{i+d-1} \left\{ \begin{array}{l} X : X \rightarrow YZ \text{ is a rule,} \\ Y \in S_{ij}, Z \in S_{j+1,i+d} \end{array} \right\}$$

Complexity: $\mathcal{O}(n^3)$.

Of what does this triply nested loop remind you?

Of what does this triply nested loop remind you?

- Matrix Multiplication
- In fact, better matrix multiplication algorithms yield (asymptotically) better general context free parsing algorithms
- Strassen's algorithm requires $O(n^{2.81})$ instead of $O(n^3)$ multiplications

Summary of Context-Free Recognition

- CFL to PDA reduction yields nondeterministic automaton
- By use of Chomsky Normal Form and dynamic programming, there is a general $O(n^3)$ non-stack-based algorithm
- The deterministic CFLs are the languages recognizable by deterministic PDAs
- E.g. $\{wcw^R : w \in \{a,b\}^*\}$ is a deterministic CFL but $\{ww^R : w \in \{a,b\}^*\}$ (even palindromes) is not
- Methods used in compliers are deterministic stack-based algorithms, requiring that the source language be deterministic CF or a special type of deterministic CF (LR(k), etc.).

Beyond Context-Free Languages

- A **Context-Sensitive Grammar** allows rules of the form $\alpha \to \beta$, where α and β are strings and $|\alpha| \leq |\beta|$, so long as α contains at least one nonterminal.

- The possibility of using rules such as $aB \to aDE$ makes the grammar "sensitive to context"

- Is there an algorithm for determining whether $w \in L(G)$ where G is a CSG?

- But the field moved, and now we also move, from syntactic structures to computational difficulty