Lecture 10: Algorithms for CFL's

CSCI 81
Spring, 2012

Kim Bruce

Algorithms for CFL's

- Given a cfg, G, and w in Σ^*, is $w \in L(G)$?
- Given a cfg, G, is $L(G) = \emptyset$?
- Given a cfg, G, is L(G) infinite?
- All are decidable!

Is $w \in L(G)$?

- Convert G to CNF G'
- If $w = \varepsilon$, see if $S \rightarrow \varepsilon$ is in rules.
- Otherwise look at all derivations of length $\leq 2\,|w| - 1$. If not there, then not in language.
- How efficient? Let $|w| = n$
  - $|R|^{2n-1}$ derivations, each of length $2n-1$. Thus $O(n\,2^n)$
  - With work (see later), can find $O(n^3)$ algorithm.
    - Want O(n)!!

Thinning cfg

- Say non-terminal V is non-productive if there is no string $w \in \Sigma^*$ s.t. $V \Rightarrow^* w$.
- Algorithm: Set G' = G
  - Mark every terminal in G' as productive
  - Until entire pass through R w/no marking
    - For each $X \rightarrow \alpha$ in R: If every symbol in α marked as productive, but X not yet marked productive, then mark it as productive
  - Remove from $V_{G'}$ all non-productive symbols
  - Remove from $R_{G'}$ all rules w/non-productive symbols on left or right side.

Algorithm for Emptiness

- Run algorithm to mark non-productive symbols. If S non-productive then L(G) = ∅.

Algorithm for Finiteness

- Let G by cfg. Use proof of pumping lemma.
  - Let G' be equivalent grammar in CNF.
  - Let n = #non-terminals. Let $k = 2^{n+1}$.
  - If there is a w ∈ L(G) s.t. |w| > k then can pump, so ∞
  - Claim if L(G) ∞ then exist w ∈ L(G) s.t. k< |w| ≤ 2k.
    - Spose fails. Then ∞, so let w' ∈ L(G) be shortest s.t. |w'| > 2k.
    - Pump with i = 0 to get shorter. But |vxy| < k & thus |vy| < k.
    - Thus uxy ∈ L(G), |uxy| < |w'|, but |uxy| > k. Contradiction to assumption w' shortest!
    - Thus L(G) is ∞ iff exists w ∈ L(G) s.t. k < |w| ≤ 2k

Parsing CFL's

- Created non-deterministic PDA.
  - Backtracking computation hard to get right.
  - Having to backtrack on input painful
- More efficient dynamic programming
- Later see deterministic language better

CYK Algorithm

- Cocke–Younger–Kasami mid-60's
- Uses dynamic programming to save partial results rather than recalculate each time.
  - $O(n^3)$
- Compare recursive fibonacci w/ iterative
  - Bottom-up rather than top-down
  - Record in advance rather than memoization

CYK

- Convert cfg G to G' in Chomsky Normal Form
- Let $w = w_1 \ldots w_n$ be string to be parsed. Define $\alpha(i,j)$ to be $\{B \mid B \Rightarrow^* w_i \ldots w_j\}$
  - So $x \in L(G)$ iff $S \in \alpha(1,n)$
  - Key idea: What non-terminals give substrings?
- Recursive definition:
  - $\alpha(i,i) = \{C \mid C \rightarrow w_i\}$
  - $\alpha(i,k) = \{C \mid C \rightarrow AB \;\&\; A \in \alpha(i,j) \wedge B \in \alpha(j+1,k)$ for some $j\}$

Fill in Table

$\alpha(1,1)$	$\alpha(1,2)$	$\alpha(1,3)$	…	$\alpha(1,n-1)$	$\alpha(1,n)$
	$\alpha(2,2)$	$\alpha(2,3)$	…	$\alpha(2,n-1)$	$\alpha(2,n)$
		$\alpha(3,3)$	…	$\alpha(3,n-1)$	$\alpha(3,n)$
			…	…	…
				$\alpha(n-1,n-1)$	$\alpha(n-1,n)$
					$\alpha(n,n)$

Each entry computed from entries in same row & column:

$\alpha(1,3)$ from $\alpha(1,1)$ & $\alpha(2,3)$, $\alpha(1,2)$ & $\alpha(3,3)$, etc.

Slide across row and down column.

Why $O(n^3)$?

Using CYK

- Let G be grammar for balanced parens in CNF:
  - $S \rightarrow SS$, $S \rightarrow LT$, $S \rightarrow LR$
  - $T \rightarrow SR$
  - $L \rightarrow ($, $R \rightarrow)$
- Parse ( ) (( ))
- Generally most entries are empty
- What if two entries in same slot?
  - Better to store rule rather than just left-hand side.