Algorithms for CFLs

Thursday, February 25, 2021 11:41 AM

$\Sigma = \{a, b\}$

$L = \{ ww \mid w \in \Sigma^* \}$ → **not** a CFL

$\overline{L}$ all words that are not in L

examples : $abba \in \overline{L}$, $aaa \in \overline{L}$

$\varepsilon \in L$, $abab \in L$, $abaaba \in L$

I will describe a PDA informally

A word is in $\overline{L}$ if

(i) it has odd length → can be checked with a DFA

OR (ii) it has even length and there is a pair of corresponding positions with different letters.

$\underbrace{x_1 x_2 \cdots x_{i-1}}_{(i-1)\text{ letters}}$ (x_i) $\underbrace{x_{i+1} \cdots x_n}_{(n-i)}$ $\underbrace{y_1 \cdots y_{i-1}}_{(i-1)}$ (y_i) $\underbrace{y_{i+1} \cdots y_n}_{n-i}$

x_i & y_i are both in position i of the two halves of the word.

The PDA will guess which pair of positions is mismatched.

It reads $x_1, x_2, \cdots x_{i-1}$ and pushes them on the stack. Then it memorises x_i. Then it has to find y_i; keep reading $x_{i+1}, x_{i+2}, \cdots$ upto x_n and pop the stack. When the stack is empty start pushing symbols onto the stack. Now guess where y_i is : compare y_i with the memorized x_i → if they are the same reject. If $x_i \neq y_i$ we need to verify that they were really at corresponding positions. To check this we read the rest of the word and pop the stack as we go. If the word ends exactly when the stack becomes empty x_i & y_i really were at corresponding positions.

— x —

Is there a CFG for $\overline{L}$?

$V = \{S, A, B, C\}$ $T = \Sigma = \{a, b\}$

$S \to AB \mid BA \mid A \mid B$

$A \to CAC \mid a$ $B \to CBC \mid b$

$C \to a \mid b$

A produces all strings of odd length with 'a' as the middle letter

B produces - - - - with 'b' as the middle letter

$S \to AB$ and $S \to BA$ produce even length strings

$A \xrightarrow{*} xay$ $|x| = |y| = n$

$B \xrightarrow{*} ubv$ $|u| = |v| = m$

$S \to AB \xrightarrow{*} xayubv$

a is at position $n+1$

b is at position $2n+m+2$

Total length $2n+2m+2$

$\underbrace{\qquad}_{n+m+1} \mid \underbrace{\qquad}_{n+m+1}$

→ a at $n+1$, then $n+m+1$ further → b at $2n+m+2$

Hence the a & b are guaranteed to be at corresponding positions. The two halves are guaranteed to be different so the word $\in \overline{L}$.

An algorithm to determine whether $w \in L(G)$

w → given a word

G → a given context free grammar

We will assume G is in Chomsky normal form

$A \to BC$ or $A \to a$

Alg is $O(n^3)$ where n is $|w|$.

DYNAMIC PROGRAMMING

Given $G = (V, \Sigma, P, S)$

Input $w = a_1 \cdots a_n$, each $a_i \in \Sigma$

Work bottom-up to construct partial derivations of pieces of w & then eventually all of w.

Inductively define a 2-indexed family of subsets of V.

$j \geq i$ $X_{ij} := \{ A \in V \mid A \xrightarrow{*} a_i \cdots a_j \}$

BASE $X_{ii} = \{ A \in V \mid A \to a_i \}$ $X_{11}, X_{22}, X_{33}, \ldots$

Then I construct $X_{12}, X_{23}, X_{34}, \ldots$ one row

Then I construct $X_{13}, X_{24}, X_{35}, \ldots$ next row

We will put all these sets in a table each of these will be in a row

X_{ij} will be in row $(j-i)+1$

When I want to compute X_{ij} I will know X_{ik} & X_{kj} for all $i \leq k \leq j$

If $B \in X_{ik}$ & $C \in X_{(k+1)j}$ and $A \to BC$ is a production rule then $A \in X_{ij}$

$B \xrightarrow{*} a_i \cdots a_k$ $C \xrightarrow{*} a_{(k+1)} \cdots a_j$

$A \to BC \xrightarrow{*} a_i \cdots a_j$

In the end we want to know $S \in X_{1n}$?

$S \to AB \mid BC$ $A \to BA \mid a$ $B \to CC \mid b$

$C \to AB \mid a$

Want to know $baaba \in L(G)$?

5	{A, S} (X_{15})				
4	∅	{S, A, C}			
3	∅	{B} (X_{24})	{B}		
2	{A, S}	{B}	{S, C}	{A, S}	
1	{B}	{A, C}	{A, C}	{B}	{A, C}
	b	a	a	b	a

We see that $S \in X_{15}$ so starting from S we can generate the whole word.

TAKEAWAY: There is an $O(n^3)$ alg to answer the question "is $w \in L(G)$".

Best known alg is $O(n^{2.8})$.

I will NEVER ask you to execute this algorithm on a test.

There are decision procedures to test

- if $L(G) = \emptyset$
- if $L(G)$ is finite or infinite

What about $L(G) = \Sigma^*$?

THERE IS NO ALGORITHM TO DECIDE THIS!