# Algorithms for CFLs

Thursday, February 25, 2021 11:41 AM

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

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

$\overline{L}$ all words that are <u>not</u> in $L$

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

$\varepsilon \in L$, $ababab \in L$, $abaaba \in L$

I will describe a PDA <u>informally</u>

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

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

<u>OR</u> (ii) it has even length <u>and</u> there is a pair of <u>corresponding positions</u> 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 <u>guess</u> 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.

$\overline{L} \times \overline{L}$

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

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

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

$A \rightarrow CAC \mid a \quad B \rightarrow CBC \mid b$

$C \rightarrow a \mid b$

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

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

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

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

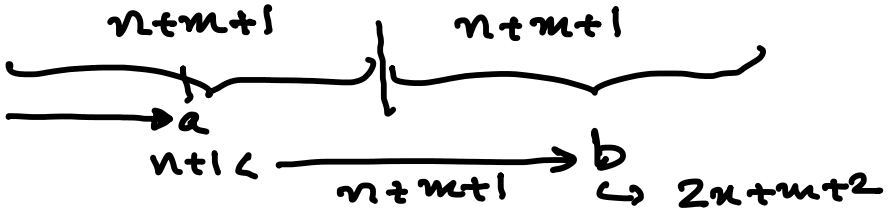
$B \xrightarrow{*} ubv \qquad |u| = |v| = m$

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

a is at position $n+1$

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

Total length $2n+2m+2$




Hence the a & b are guaranteed to be at corresponding positions.

The two halves are guaranteed to be different so the word $\in L$.

An <u>algorithm</u> to determine whether

$w \in L(G)$

$w \rightarrow$ given a word

$G \rightarrow$ a given context free grammar

We will assume $G$ is in Chomsky normal form

$$A \rightarrow BC \quad \text{or} \quad A \rightarrow a$$

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

DYNAMIC PROGRAMMING

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

<u>Input</u> $w = a_1 \cdots a_n$, 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 $V$.

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

<u>BASE</u> $X_{ii} = \{ A \in V \mid A \rightarrow a_i \}$ $\quad X_{11}, X_{22}, X_{33} \cdots$

Then I construct $X_{12}, X_{23}, X_{34} \cdots$ 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 $i \le k \le j$

If $B \in X_{ik}$ & $C \in X_{(k+1)j}$

and $A \to BC$ is a production rule

Then $A \in X_{ij}$

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

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

In the end we want to know

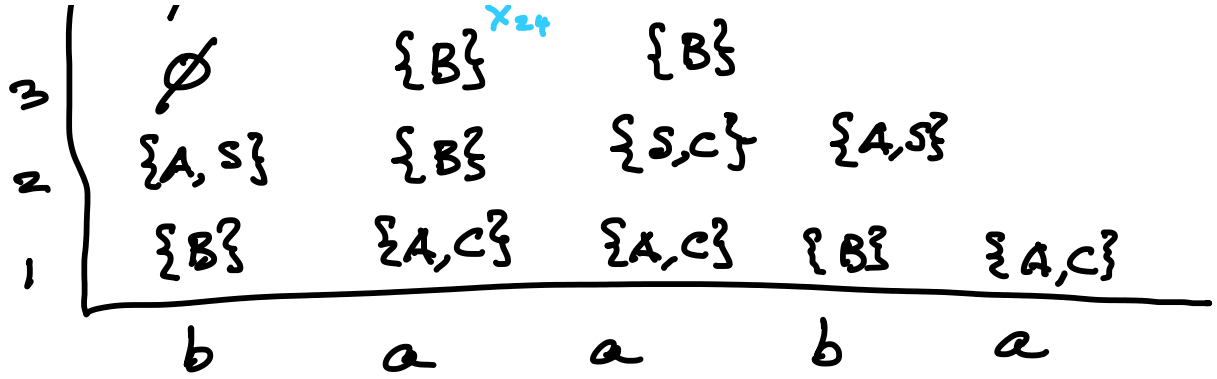
$$S \in X_{1n} \; ?$$

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

$C \to AB \mid a$

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

| | |
|---|---|
| 5 | |
| 4 | $\emptyset$ $\{S, A$ |


3
2
1
∅
{B}
×24
{B}
{A, S}
{B}
{S, C}
{A, S}
{B}
{A, C}
{A, C}
{B}
{A, C}
b
a
a
b
a