17/02/2022

Theory of Computation

Pumping Lemma for Regular Languages :

Pumping Lemma :

L is a regular Lang. Then there exists a constant $n \geq 1$ (which depends on L) such that for every string $w \in L$, such that $|w| \geq n$, we can break w into three substring $w = xyz$ such that

$y \neq \epsilon$

$|xy| \leq n$

$\forall k \geq 0$,

Proof :

- L is regular, L is accepted by deterministic finite automaton M.
- Suppose that n is the number of states of M, and let w be a string of length n or greater.
- Consider now the first n steps of computation of M on w :

$$\delta(q_0, w_1 \dots w_n) = \delta(q_1, w_2 \dots w_n)$$
$$= \delta(q_3, w_3 \dots w_n)$$
$$\vdots$$
$$\delta(q_n, \epsilon)$$

q_0 = initial state

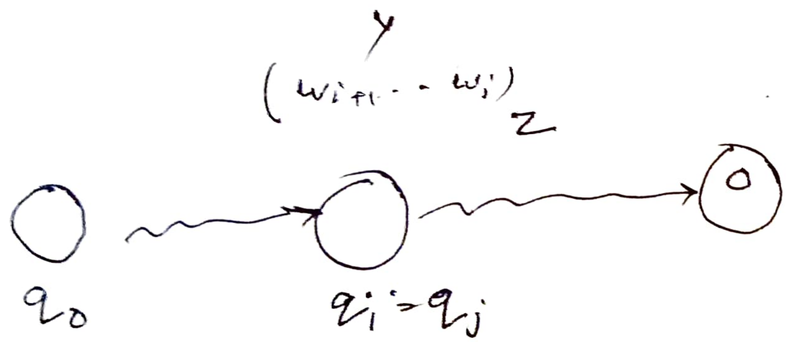
• M has only n states and there are $(n+1)$ configurations $(q_i, w_{i+1} \cdots w_n)$ appearing above. By Pigeon hole Principle, there exists i and j. $0 \le i < j \le n$ such that $q_i = q_j$.

• string $y = w_i w_{i+1} \cdots w_j$ drives M from state q_i back to state q_i and this string 'y' is non-empty, since $i < j$.

• Either,

Removing this substr from w or repeating any number of times in w, just after j^{th} symbol.

M accepts $xy^k z \in L$, for each k



Example 1 : $L\{a^i b^i \mid i \geq 0\}$ is not regular.

- Suppose L is regular, pumping lemma holds true for some $n \in I$.
- Let $w = a^n b^n \in L$
- w can be rewritten as $w = xyz$ such that $|xy| \leq n$ and $y \neq \epsilon$, that is, $y = a^i$ for some $i > 0$.
- $xyz = a^{n-1} b^n \notin L$, contradicting the theorem. L is not regular.

Example 2 : $L = \{a^n : n$ is prime$\}$ is not regular.

Let $w = xyz$, $x = a^p$, $y = a^q$, $z = a^r$ $p, r \geq 0$ and $q > 0$.

Theorem, $xy^n z \in L$ for each $n \geq 0$; that is $p + nq + r$ is prime, for each $n \geq 0$

This is impossible,

$n = p + 2q + r + 2$,

then $p + nq + r = (q+1)(2q + r + p)$ which is not prime.

Closure Properties of Regular set:

- Two regular exp. L_1 and L_2 over Σ are closed under union operation.

- Complement of regular exp. is regular.

Time Complexity and FA

① There is an exponential ~~people~~ algo, which NFA constructs an equivalent DFA.

② There is a polynomial algo which, gives a regular expression

Context - Free Grammar [CFG]

A CFG is a 4-tuple $G = \{V_N, \Sigma, S, P\}$ where

V_N and Σ : Disjoint sets called set of non terminal and terminal (or alphabets) symbols

S – start symbol

P : Grammar rules or productions, a finite set of formulae of form $A \to \alpha$, where $A \in V_N$, and $\alpha \in (V_N \cup \Sigma)^*$

CFG Examples:

① Language 'pal' of Palindromes over $\{a, b\}$

→ $\epsilon, a, b \in$ pal

→ For any $S \in$ pal, aSa and bSb are in pal.

→ No other strings in pal

→ $S \to \epsilon \mid a \mid b \mid aSa \mid bSb$

② Compliment of Palindrome.

• aAb and bAa are in N,

• $S \in N$, aSa and bSb in N

∴ $S \to e \mid Aa \mid Ab$

① Construct a CFG for $L = (011+1)^*(01)^*$

②

→ $\epsilon \in L$

→ $0x1$, $1x0$

$S \to \epsilon \mid 0S1 \mid 1S0$

Theorem: If L_1 and L_2 are CFL's, then the languages $L_1 \cup L_2$, $L_1 \cdot L_2$ and L_1^* are also CFLs.

Proof:

- Let us consider two Grammars $G_1 = (V_1, \Sigma, S_1, P_1)$ and $G_2 = (V_2, \Sigma, S_2, P_2)$ generating L_1 and L_2 respectively.
- We show how to construct a new CFG for each of these 3 cases.

$G_4 = (V_4, \sigma, S_4, P_4)$ generating $L_1 \cup L_2$.

$L = \{0^i 1^j 0^k \mid j > i+k\}$

$[0^i 1^i][1^m][1^k 0^k]$, $m = j-i-k$

$L_1 : \{0^i 1^i \mid i >= 0\} \Rightarrow A \quad 0A1 \mid \epsilon$

$L_2 : \{1^m \mid m > 0\} \quad B \Rightarrow 1B \mid 1$

$L_3 : \{1^k 0^k \mid k \geq 0\} \quad C \Rightarrow 1C0 \mid \epsilon$

$L = L_1 L_2 L_3$

$\{S \to ABC,\ A \to 0A1 \mid \epsilon\ \ B \to 1B \mid 1\ \ C \to 1C0 \mid \epsilon\}$