Regular Languages are in Nick's Class

We give an $\mathcal{NC}$ algorithm which decides the mambership problem for a regular language, proving that the class of regular languages is a subclass of Nick's Class.

Boolean Matrices

A Boolean matrix is a matrix whose entries are of Boolean type. We write 1 for true and 0 for false.

Boolean Matrix Operations Operations, such as addition and multipication, on Boolean matrices are similar to operations on number matrices, except that disjunction and conjunction replace addition and multiplication. That is, number matrix operations use $(\times, +)$ algebra, while Boolean matrices use (and, or) algebra. For examle:

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Transition Matrices

Let $L \subseteq \Sigma^*$ be a regular language over Σ, and let $M = (\Sigma, Q, F, q_0, \delta)$ be an NFA which accepts L. Let $Q = \{q_i : 0 \leq i < k\}$. For any $a \in \Sigma$ we define the *transition matrix* T_a to be the $k \times k$ Boolean matrix where

$$T_a[i, j] = \begin{cases} 1 \text{ if } q_j \in \delta(a, q_i) \\ 0 \text{ otherwise} \end{cases} \quad \text{for all } \ 0 \leq i, j < k$$

We extend this definition by mapping concatenation to matrix multiplication, *i.e.,* $T_{uv} = T_u T_v$, and T_λ is the identity matrix.

Algorithnm $\mathcal{A}$

$\mathcal{A}$ reads a string $w \in \Sigma^*$ and returns 1 if and only if $w \in L$. Let $n = |w|$. We first assume $n = 2m$. If n is not a power of 2, we pad w with empty strings to make its length a power of 2. For example, if $w = abccbcabbabcc$, of length 13, we rewrite w as abccbcabbabcc$\lambda\lambda\lambda$ of length 16, making $m = 4$. For any $0 \leq p \leq m$, w is the concatenation of 2^{m-p} substrings of length 2^p. Let $\mathcal{S}$ be the set of all substrings thus obtained. $\mathcal{A}$ is as follows:

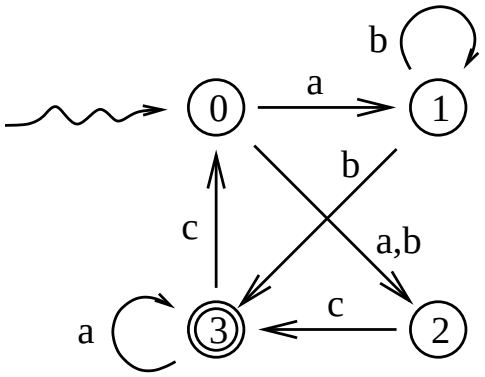
Algorithm $\mathcal{A}$

Compute T_λ, the identity matrix.
Compute the transition matrix T_a for all $a \in \Sigma$.
For all p from 1 to m
  For all $u \in \mathcal{S}$ of length 2^p
    Let $u = xy$ where $|x| = |y| = 2^{p-1}$
    $T_u = T_x T_y$
If ($T_w[0, f]$ for some $q_f \in F$) return 1. ($w \in L$)
Else return 0. ($w \notin L$)

Since we are taking the sizes of Σ and Q to be $O(1)$, all T_a for $a \in \Sigma$ can be computed in $O(1)$ time by one processor. The remainder of the algorithm consists of $n - 1$ matrix multiplications which can be done in $O(\log n)$ time with $O(n)$ processors; $\mathcal{A} \in \mathcal{NC}$.

Example

Let $\Sigma = \{a, b, c\}$ and $L = L(M)$, where M is the followinhg NFA. Let $w = acacabba$.



We compute transition matrices of elementary strings, then copy to the 8 leaves of our computation tree. Each matrix in rows 2–4 is the product of the two above it. Then $w \in L$ since $T_w[0,3] = 1$ and $q_3 \in F$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_\lambda \qquad \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_a \qquad \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad T_b \qquad \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad T_c$$

$$\begin{bmatrix} 0&1&1&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&1 \end{bmatrix} T_a \quad \begin{bmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&1 \\ 1&0&0&0 \end{bmatrix} T_c \quad \begin{bmatrix} 0&1&1&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&1 \end{bmatrix} T_a \quad \begin{bmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&1 \\ 1&0&0&0 \end{bmatrix} T_c \quad \begin{bmatrix} 0&1&1&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&1 \end{bmatrix} T_a \quad \begin{bmatrix} 0&0&1&0 \\ 0&1&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} T_b \quad \begin{bmatrix} 0&0&1&0 \\ 0&1&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} T_b \quad \begin{bmatrix} 0&1&1&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&1 \end{bmatrix} T_a$$

$$\begin{bmatrix} 0&0&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \\ 1&0&0&0 \end{bmatrix} T_{ac} \quad \begin{bmatrix} 0&0&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \\ 1&0&0&0 \end{bmatrix} T_{ac} \quad \begin{bmatrix} 0&1&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} T_{ab} \quad \begin{bmatrix} 0&0&0&0 \\ 0&0&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} T_{ba}$$

$$\begin{bmatrix} 1&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&1 \end{bmatrix} T_{acac} \qquad \begin{bmatrix} 0&0&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} T_{abba}$$

$$\begin{bmatrix} 0&0&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} T_{acacabba} = T_w$$

Exercise 1: Let M be the NFA with transition matrix

.

	a	b
0	$0, 1$	$\emptyset$
1	$\emptyset$	$1, 2$
2	0	0

Use the tournament method shown above to determin whether $aabab \in L(M)$.