Harvard CS 121 and CSCI E-207

Lecture 5: NFAs and DFAs Closure Properties

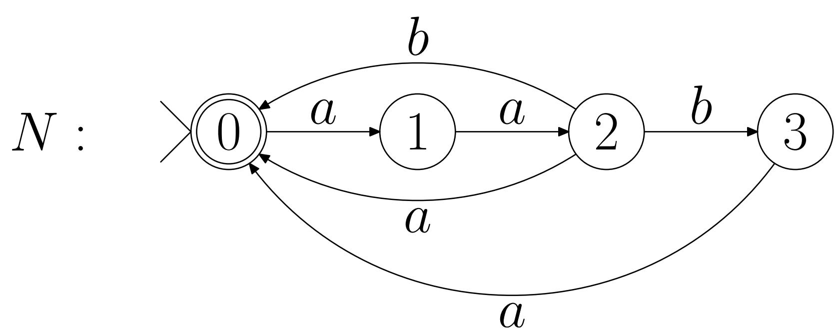
Harry Lewis

September 16 2010

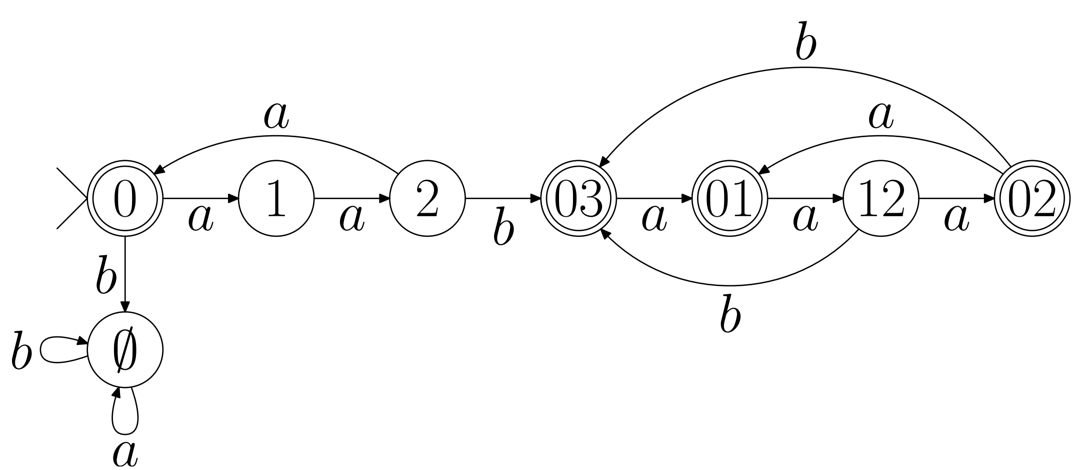
Reading: Sipser, §1.2.

Example of the SUBSET CONSTRUCTION

NFA N for $\{x_1 x_2 \cdots x_k : k \geq 0$ and each $x_i \in \{aab, aaba, aaa\}\}$.



N starts in state 0 so we will construct a DFA M starting in state $\{0\}$. Here it is:



All other transitions are to the “dead state” $\emptyset$. The other states are unreachable, though technically must be defined. Final states are all those containing 0, the final state of M.

Formal Construction of DFA M from NFA $N = (Q, \Sigma, \delta, q_0, F)$

On the assumption that $\delta(p, \varepsilon) = \emptyset$ for all states p.

(i.e., we assume no ε-transitions, just to simplify things a bit)

$M = (Q', \Sigma, \delta', q_0', F')$ where

$$
\begin{aligned}
Q' &= P(Q) \\
q_0' &= \{q_0\} \\
F' &= \{R \subseteq Q : R \cap F \neq \emptyset\} \text{ (that is, } R \in Q'\text{)} \\
\delta'(R, \sigma) &= \{q \in Q : q \in \delta(r, \sigma) \text{ for some } r \in R\} \\
&= \bigcup_{r \in R} \delta(r, \sigma)
\end{aligned}
$$

Nondeterminism gives us a new programming tool

- Strings that begin with $aaba$

Nondeterminism gives us a new programming tool

- Strings that begin with $aaba$

- Strings that end with $aaba$

Nondeterminism gives us a new programming tool

- Strings that begin with $aaba$

- Strings that end with $aaba$

- Strings that begin or end with $aaba$

Nondeterminism gives us a new programming tool

- Strings that begin with $aaba$
- Strings that end with $aaba$
- Strings that begin or end with $aaba$
- Strings that have $aaba$ as a substring anywhere

Closure Properties

Theorem: The class of regular languages is closed under:

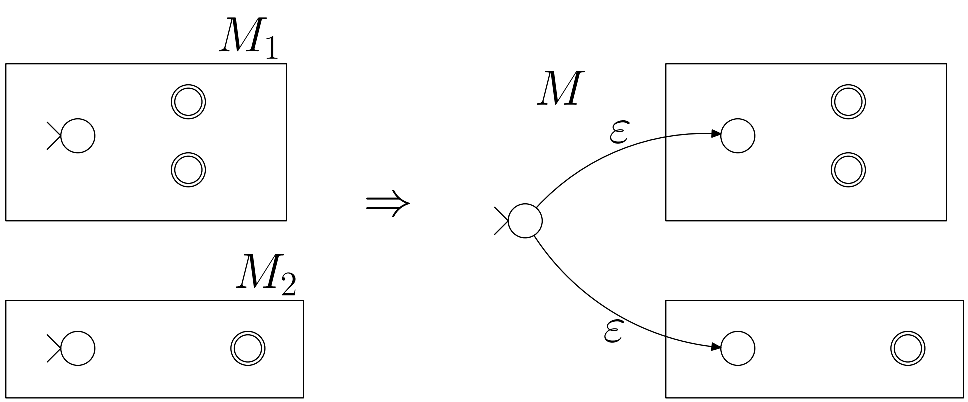
- Union: $L_1 \cup L_2$
- Concatenation: $L_1 \circ L_2 = \{xy : x \in L_1 \text{ and } y \in L_2\}$
- Kleene *: $L_1^* = \{x_1 x_2 \cdots x_k : k \geq 0 \text{ and each } x_i \in L_1\}$
- Complement: $\overline{L_1}$
- Intersection: $L_1 \cap L_2$

Closure Properties

Theorem: The class of regular languages is closed under:

- Union: $L_1 \cup L_2$
- Concatenation: $L_1 \circ L_2 = \{xy : x \in L_1 \text{ and } y \in L_2\}$
- Kleene *: $L_1^* = \{x_1 x_2 \cdots x_k : k \geq 0 \text{ and each } x_i \in L_1\}$
- Complement: $\overline{L_1}$
- Intersection: $L_1 \cap L_2$

Union: If L_1 and L_2 are regular, then $L_1 \cup L_2$ is regular.



M has the states and transitions of M_1 and M_2 plus a new start state ε-transitioning to the old start state

Concatenation, Kleene *, Complementation

Concatenation:

$$L(M) = L(M_1) \circ L(M_2)$$

Kleene *:

$$L(M) = L(M_1)^*$$

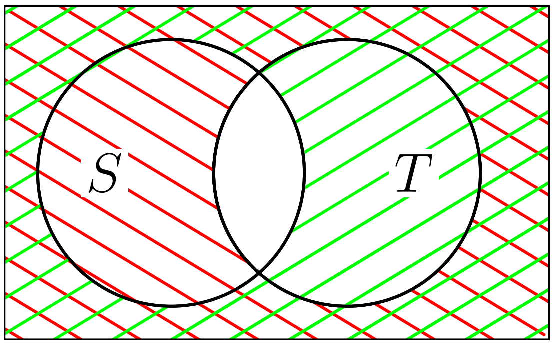
Complement:

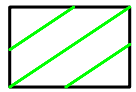
$$L(M) = \overline{L(M_1)}$$

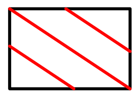
Closure under Intersection

Intersection

$$S \cap T = \overline{\overline{S} \cup \overline{T}}$$



 $= \overline{S}$

 $= \overline{T}$

Hence closure under union and complement implies closure under intersection

A more constructive and direct proof of closure under intersection

Better way ("Cross Product Construction"):

From DFAs $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, construct $M = (Q, \Sigma, \delta, q_0, F)$:

$$\begin{aligned} Q &= Q_1 \times Q_2 \\ F &= F_1 \times F_2 \\ \delta(\langle r_1, r_2 \rangle, \sigma) &= \langle \delta_1(r_1, \sigma), \delta_2(r_2, \sigma) \rangle \\ q_0 &= \langle q_1, q_2 \rangle \end{aligned}$$

Then $L(M_1) \cap L(M_2) = L(M)$

Some Efficiency Considerations

The subset construction shows that any n-state NFA can be implemented as a 2^n-state DFA.

NFA States	DFA States
4	16
10	1024
100	2^{100}
1000	$2^{1000} \gg$ the number of particles in the universe

How to implement this construction on ordinary digital computer?

NFA states

$1, \ldots, n$

DFA state bit vector

$$\begin{array}{|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 0 & \cdots & 1 \\ \hline \end{array}$$
$$\begin{array}{cccccc} 1 & 2 & & & & n \end{array}$$

Is this construction the best we can do?

Could there be a construction that always produces an n^2 state DFA for example?

Theorem: For every $n \geq 1$, there is a language L_n such that

1. There is an $(n+1)$-state NFA recognizing L_n.
2. There is no DFA recognizing L_n with fewer than 2^n states.

Conclusion: For finite automata, nondeterminism provides an *exponential savings* over determinism (in the worst case).

Proving that exponential blowup is sometimes unavoidable

(Could there be a construction that always produces an n^2 state DFA for example?)

Consider (for some fixed n=17, say)

$$L_n \quad = \quad \{w \in \{a,b\}^* : \text{the } n\text{th symbol from the right end of } w \text{ is an } a\}$$

- There is an $(n+1)$-state NFA that accepts L_n.
- There is no DFA that accepts L_n and has $< 2^n$ states

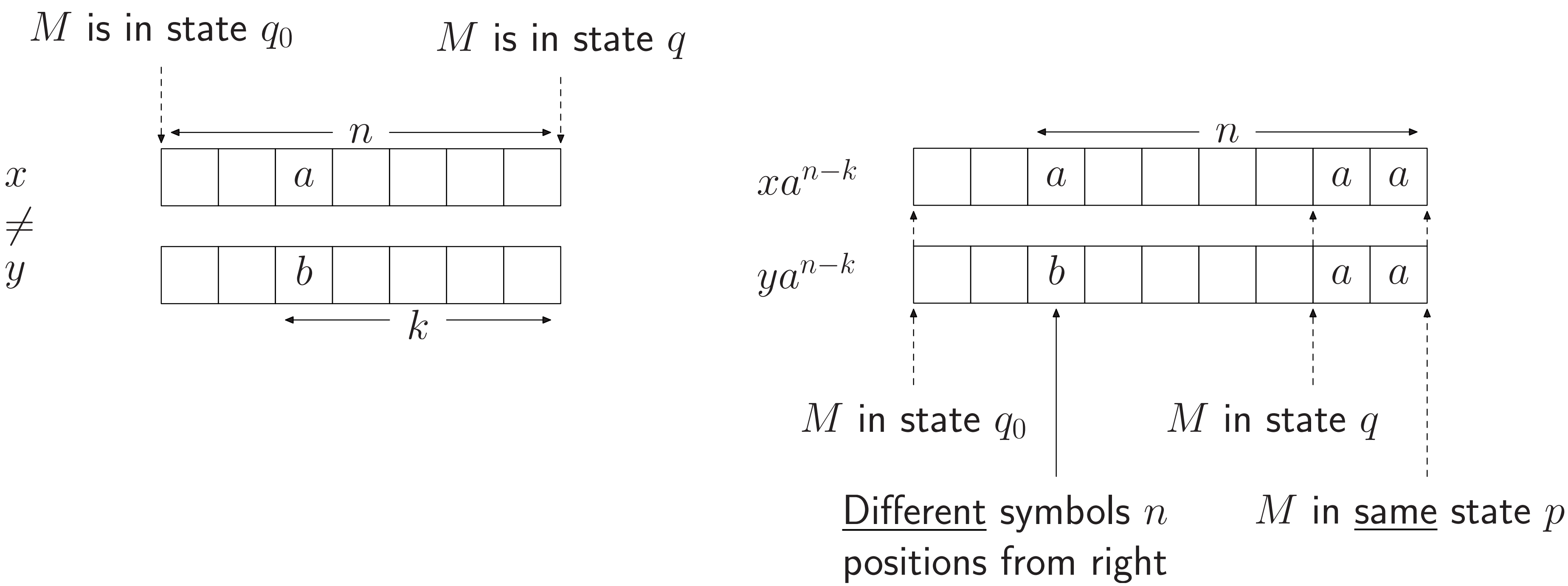
A "Fooling Argument"

- Suppose a DFA M has $< 2^n$ states, and $L(M) = L_n$
- There are 2^n strings of length n.
- By the pigeonhole principle, two such strings $x \neq y$ must drive M to the same state q.
- Suppose x and y differ at the k^{th} position from

  the right end (one has a, the other has b)

  $(k = 1, 2, \ldots, \text{or } n)$
- M must treat xa^{n-k} and ya^{n-k} identically (accept both or reject both). These strings differ at position n from the right end.
- So $L(M) \neq L_n$, contradiction. QED.

Illustration of the fooling argument



- x and y are different strings

  (so there is a position k where one has a and the other has b)

- But both strings drive M from s to the same state q

What the argument proves

- This shows that the subset construction is within a factor of 2 of being optimal

- In fact it is optimal, i.e., as good as we can do in the *worst case*.

- Still, in many cases, the “generate-states-as-needed” method yields a DFA with $\ll 2^n$ states

  (e.g. if the NFA was deterministic to begin with!)