Finite State Acceptors (finite automaton)

Def: A Moore machine is a finite state acceptor if

(1) there is a distinguished state called the initial state q_0.

(2) $Y = \{0,1\}$

In this case, we can define an acceptor as:

$M = (Q, \Sigma, \delta, q_0, F)$ where $\Sigma = X$ (input set)

and $F = \{ q_j \mid \lambda(q_j) = 1 \}$ set of final states.

$\delta : Q \times \Sigma \to Q$ transition function

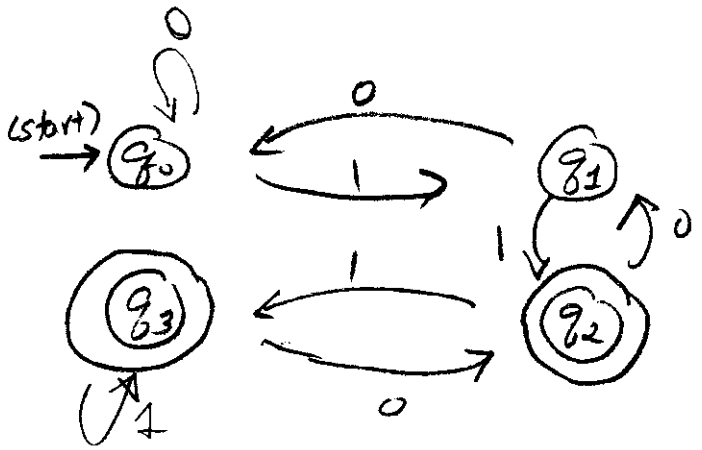
Note: The set of strings over Σ^* accepted by M is:

$L = T(M) = \{ x \mid \tilde{\delta}(q_0, x) \in F \}$ ($\tilde{\delta}$ is same as δ^*.)

Equivalently we say the language L accepted by M is $L = L(M) = \beta_{q_0}^{-1}(1)$ for the Moore representation.

Note: L is a finite state language $\Leftrightarrow$ it has a finite state acceptor.

Example



$L(M) = ?$

Note: final states have double circles.

Example Synthesizing a f.s.a.

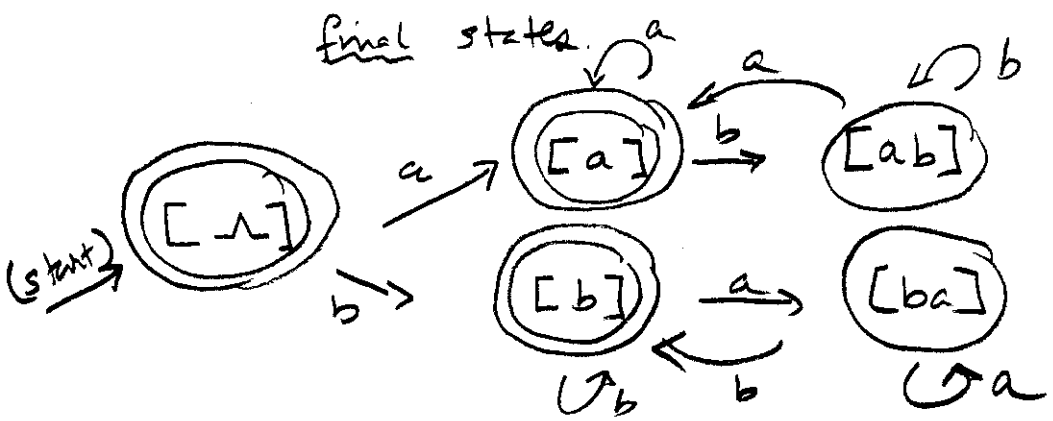
$L = \{ x \in X^* \mid x \text{ has same first and last symbol} \}$

Define $f: X^* \to \{0,1\}$ by $f(x) = 1$ if $x \in L$
$f(x) = 0$ otherwise

$\therefore$

Consider Nerode equivalence classes:

$[\Lambda]$, $[a]$, $[ab]$, $[b]$, $[ba]$



Example

$L = \{ 0^{n_1} / {}^{n_2} \mid n \geq 1 \}$ is not a finite state language.

Proof If it was finite state for some $n_1 \neq n_2$

$0^{n_1} \equiv 0^{n_2}$ (nerode equivalent) Why?

$\therefore$ also can say $\exists$ state q_0, r s.t.

$$q_0 \xrightarrow[0^{n_2}]{0^{n_1}} r$$

$\therefore 0^{n_1} / {}^{n_1} \equiv 0^{n_2} / {}^{n_1}$ (right invariance).

but $0^{n_1} / {}^{n_1}$ is accepted and $0^{n_2} / {}^{n_1}$ is not. contradiction!

Goal: Show the equivalence of 3 different ways of describing "regular languages."

① Deterministic or non-deterministic finite state acceptor M. We call the corresponding language $L(M)$ a finite state language. (Note: text calls this regular)

② Regular grammar G. We call the corresponding language $L(G)$. This language is often called a regular language.

③ Regular expressions. We call the corresponding language (or set) a regular set

We will show: equivalence between

① finite state acceptors $\longleftrightarrow$ regular grammars

② finite state acceptors $\longleftrightarrow$ regular expressions

Thus all 3 descriptions describe the same class of languages

Regular Expressions

Informally

Let Σ be an alphabet say $\Sigma = \{a, b, c\}$

(a) Concatenation operator on strings .

$x = abbc \quad y = aba \quad x \cdot y = abbcaba$

Concatenation operation on sets of strings or Languages

$L_1 \cdot L_2 = \{ x \cdot y \mid x \in L_1, y \in L_2 \}$

(b) star operator $*$. $(ab)^* = \{ \lambda, ab, abab, ababab, \ldots \}$

$L^* = \bigcup_{i=0}^{\infty} L^i$ Note $L^+ = \bigcup_{i=1}^{\infty} L^i$

(c) Union operation $\cup$ or $+$.

$L \cup M$ or $L + M = \{ x \mid x \in L \text{ or } x \in M \}$

example $aba + (ba)^* = \{ aba, \lambda, ba, baba, \ldots \}$

Formally

Let Σ be a given alphabet.

A regular expression over Σ is defined as follows:

1. $a \in \Sigma \Rightarrow a$ is a regular expression } primitive regular expressions
2. ϕ, λ are regular expressions } primitive regular expressions
3. If P and Q are regular expressions, then so are $P+Q$, $P \cdot Q$, P^*
4. Only expressions formed by 1, 2, & 3 are regular expressions.

Note: A regular expression is a string of symbols using Σ, $+$, $\cdot$, $*$.

Examples

$\Sigma = \{0,1\}$ $\quad (01+10)^*$ $\quad (0|(11)^* + 101)^* |011 + \lambda$

$\Sigma = \{a,b,c\}$ $\quad (a+bc)^* a + ab^*$

$(a+bc)^* (c + \emptyset)$

Regular expressions denote subsets of Σ^* or a language as follows: (We use $|P|$ to denote the language or set denote by reg expression P. Also say $L(P)$.)

1. $|\Phi| = \Phi$
2. $|\lambda| = \{\lambda\}$
3. For $a \in \Sigma$, $|a| = \{a\}$
4. If $P \& Q$ are regular expressions, then:

$|P+Q| = |P| \cup |Q|$

$|P \cdot Q| = |P| \cdot |Q| = \{pq \mid p \in P, q \in Q\}$

$|P^*| = |P|^* = \bigcup_{i=0}^{\infty} |P|^i$

Two regular expressions are equivalent if they denote the same sets, ie $r_1 \equiv r_2$ if $|r_1| = |r_2|$

A set denoted by a regular expression is a <u>regular set</u>.

Kleene's Theorem: L is a finite state language iff L is a regular set.

Examples

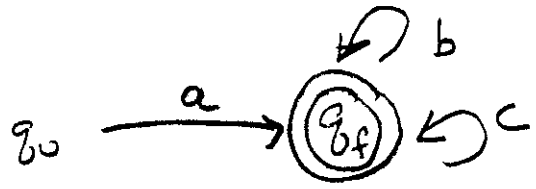
$|(a+b)^*| = \{ \lambda, a, b, aa, ab, ba, bb, aaa, \ldots \}$

$|a(b+c)^*| = \{ a, ab, ac, abb, \ldots \}$

$|(10^* + 1^* + 01)11| = \{ 111, 1011, 10011, \ldots$
$11, 111, 1111, \ldots$
$0111 \}$

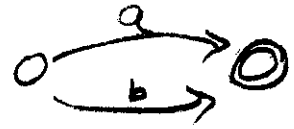
Machines?

$|a(b+c)^*|$

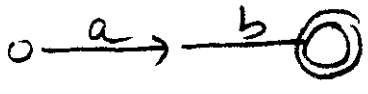


other transitions (go to a "dead" state)

Idea:



accepting $|a + b|$



accepts $|a \cdot b|$

accepts $|a^*|$

Will need notion of a nondeterministic f.s.a. to make this easier.

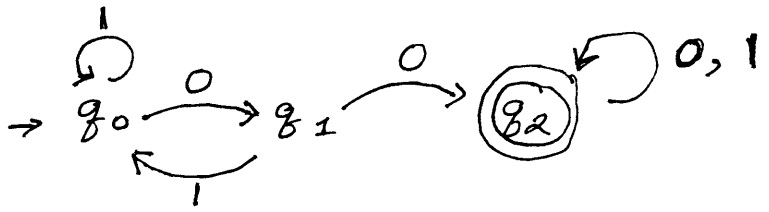
Example

Let $L = \{w \in \Sigma^* : w$ has at least one pair of consecutive zeros $\}$ $\Sigma = \{0,1\}$

Find a regular expression for L.

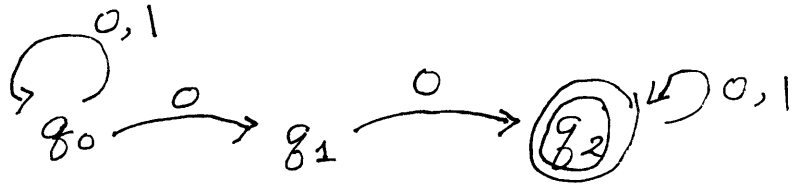
$(0+1)^* \; 00 \; (0+1)^*$

Find an f.s.a. for this language:



This is ok.

However, what if we tried the following:



This seems simpler and "closer" to the regular expression but has two next states under a 0 from state q_0!

It turns out that this can be o.k. and call this a nondeterministic f.s.a.

Note: we must define the notion of accepting a string for such a machine.

Definition An acceptor is a nondeterministic f.s.a if $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where Q, Σ, q_0, F are as before and

$$\delta : Q \times \Sigma \to 2^Q$$

That is: a state under an input can map to a subset of states of Q.

Example $\delta(q_0, a) = \{q_1, q_5, q_8\}$

Interpretation: an n.f.a. can choose any one of the next possible states.

Definition We can define an extended transition function $\tilde{\delta}$ (as before) by:

$$\tilde{\delta} : Q \times \Sigma^* \longrightarrow 2^Q$$

by

$$\tilde{\delta}(q, \lambda) = \{q\}$$
$$\tilde{\delta}(q, a) = \delta(q, a) \text{ for all } a \in \Sigma$$
$$\tilde{\delta}(q, xa) = \bigcup_{p \in \tilde{\delta}(q,x)} \delta(p, a)$$

Note We call also extend this subsets of Q by:

$$\tilde{\delta} : 2^Q \times \Sigma^* \to 2^Q$$

by

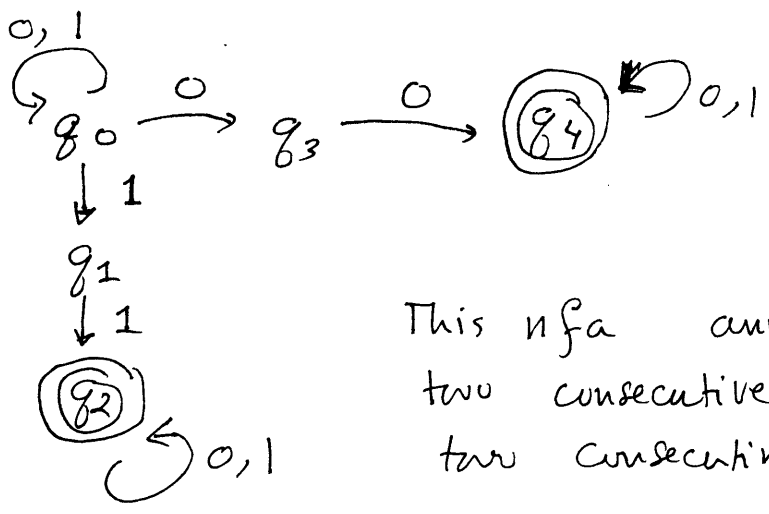
$$\tilde{\delta}(P, w) = \bigcup_{q \in P} \tilde{\delta}(q, w)$$

Definition A sentence x is accepted by an n.f.a M if $\exists$ a state $f \in F$ such that $f \in \tilde{\delta}(q_0, x)$. Equivalently $\tilde{\delta}(q_0, x) \cap F \neq \phi$

Definition $L(M) = \{ x \in \Sigma^* : \tilde{\delta}(q_0, x) \cap F \neq \phi \}$

Example $M = (\underbrace{\{q_0, q_1, q_2, q_3, q_4\}}_{Q}, \underbrace{\{0,1\}}_{X}, q_0, \underbrace{\{q_2, q_4\}}_{F}, \delta)$

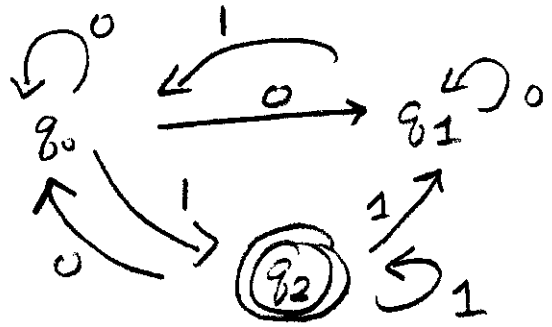
$\delta(q_0, 0) = \{q_0, q_3\}$
$\delta(q_0, 1) = \{q_0, q_1\}$
$\delta(q_1, 1) = \{q_2\}$
$\delta(q_1, 0) = \phi$
$\delta(q_2, 0) = \{q_2\}$ $\quad \delta(q_2, 1) = \{q_2\}$
$\delta(q_3, 0) = \{q_4\}$ $\quad \delta(q_3, 1) = \phi$
$\delta(q_4, 0) = \{q_4\}$ $\quad \delta(q_4, 1) = \{q_4\}$



This nfa any string with two consecutive 0's or two consecutive 1's.

Example

What language does the following n.f.a. accept?



Strings of form

0^*1

$0^*1(01)^*$

$0^*(00^*1)^*0^*1(01)^*$

etc.

Def. Two acceptors are equivalent if they accept the same language

Theorem

Let L be accepted by an n.f.a.
Then there exists a d.f.a. that accepts L.

Proof

Let $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ be an n.f.a. We define a d.f.a. $M' = \langle 2^Q, \Sigma, \delta', q_0', F' \rangle$ by

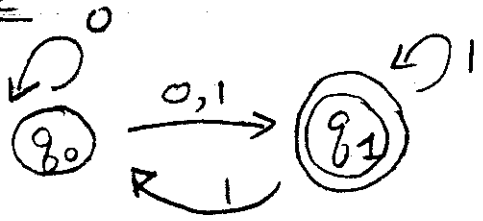
where (1) the states of M' are all subsets of states of Q.

(2) $q_0' = [q_0]$

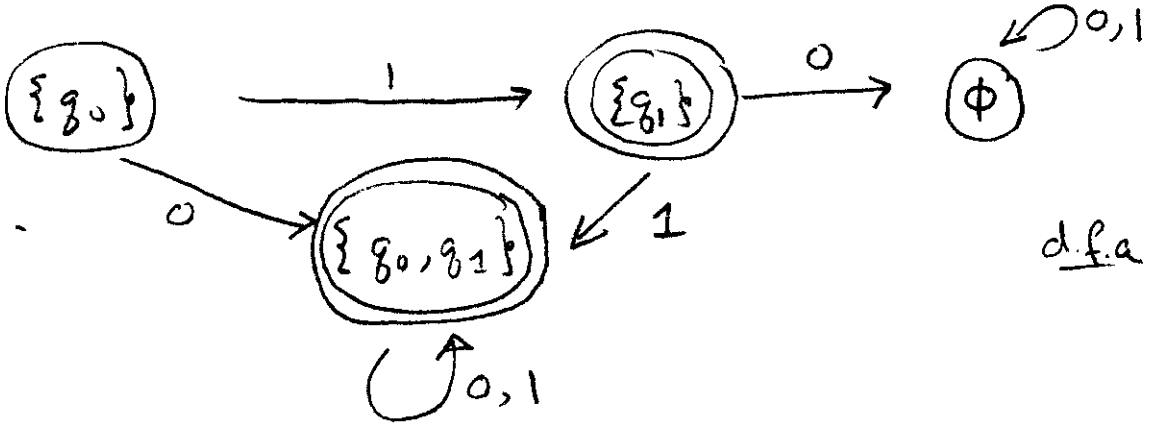
(3) $\delta'([q_1, \dots q_n], a) = [p_1, \dots p_j] \Leftrightarrow \delta(\{q_1, \dots q_n\}, a) = \{p_1, \dots p_j\}$

(4) $f' \in F'$ is a final state $\Leftrightarrow$ f' contains some $f \in F$.

Example



n.f.a



d.f.a

Proof continued.

① $\delta'(q_0', x) = [q_1, q_2, \cdots q_i]$

$\Leftrightarrow \delta(q_0, x) = \{q_1, q_2, \cdots, q_i\}$

Proof by induction on length of string x.

Basis. Show true $|x| = 0 \Rightarrow x = \lambda$.

$\therefore \delta(q_0, \lambda) = \{q_0\} \Rightarrow \delta'([q_0], \lambda) = [q_0]$.

Induction.

Assume true for inputs of length m or less. $m \geq 0$

Let xa be of length $m+1$.

$\delta'(q_0', xa) = \delta'(\delta'(q_0', x), a)$

Now $\delta'(q_0', x) = [p_1, \cdots, p_j]$

$\Leftrightarrow \delta(q_0, x) = \{p_1, \cdots p_j\}$

By definition of δ'

$$\delta'([p_1, \dots p_j], a) = [r_1, \dots, r_k]$$
$$\Leftrightarrow \delta(\{p_1, \dots p_j\}, a) = \{r_1, \dots r_k\}.$$

$\therefore$ by combining above two

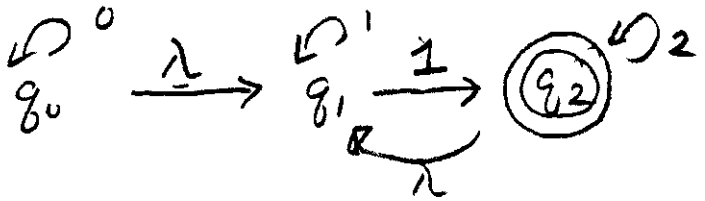
$$\delta'(q_0', xa) = [r_1, \dots r_k]$$
$$\Leftrightarrow \delta(q_0, xa) = \{r_1, \dots r_k\}.$$

(2) Furthermore, $\delta'(q_0', x)$ is in F' exactly when $\delta(q_0, x)$ contains a state of Q in F.

n.fa with $\in$-moves

Extend transitions to include empty input λ (or ϵ).

$\Sigma = \{0, 1, 2\}$



$\delta(q, a)$

	0	1	2	λ
q_0	$\{q_0\}$	Φ	Φ	$\{q_1\}$
q_1	Φ	$\{q_1, q_2\}$	Φ	Φ
q_2	Φ	Φ	$\{q_2\}$	$\{q_1\}$

$\delta : Q \times (\Sigma \cup \{\lambda\}) \to 2^Q$

Can now extend δ to $\tilde{\delta}$.

extended transition function

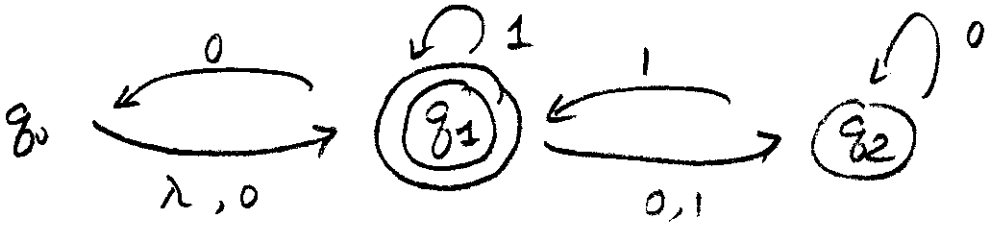
It can be proven the adding ϵ-moves does not add any power to an n.f.a. ie it still accepts the same family of languages.

Idea

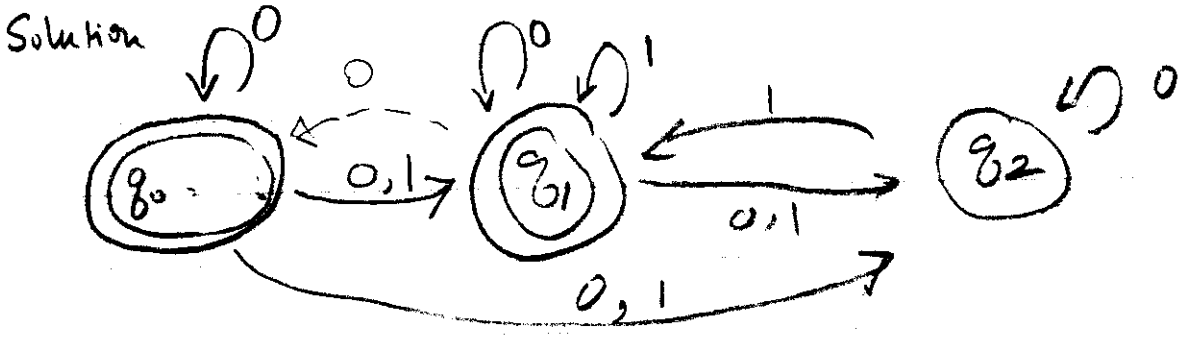
ϵ-closure. Look at at states reachable using ϵ moves only. Add these to every transition for an input a from a state q_i.

See procedure nfa- to dfa in text page 61.

Example Convert following n.f.a with ϵ moves to one without ϵ-moves.



Solution



Definition Given an acceptor $\langle Q, \Sigma, \delta, q_0, F \rangle$
Let P be a set of states of Q. Then ϵ-closure$(P) = P'$ where P' is the set of state reachable for some state $p \in P$ by a path labelled with λ's.

Can then extend δ to δ^* by

(1) $\delta^*(q, \lambda) = \epsilon$-closure$(q)$.

(2) $\delta^*(q, wa) = \epsilon$-closure$(\delta(R, a))$ for $a \in \Sigma$ and $R = \delta^*(q, w)$ for $w \in \Sigma^*$.

Note $\delta^*(q, a)$ not necessarily equal to $\delta(q, a)$.

Theorem

If a language L is accepted by an nfa with ϵ-moves, then L is accepted by an nfa without ϵ-moves.

Proof Outline

Given $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, construct
$M' = \langle Q, \Sigma, \delta', q_0, F' \rangle$, where

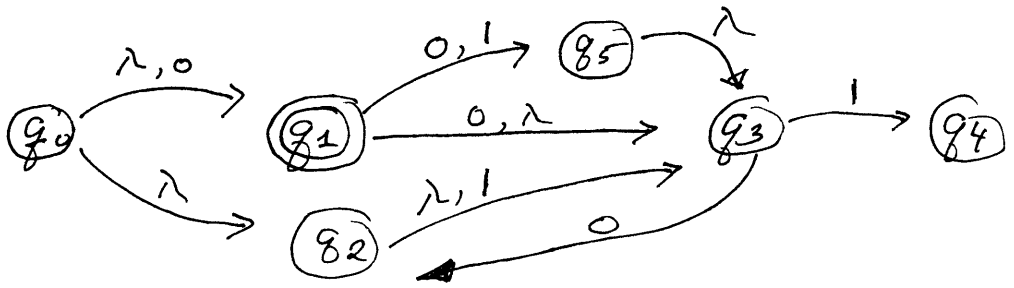
$$F' = \begin{cases} F \cup \{q_0\} & \text{if } \epsilon\text{-Closure}(q_0) \text{ contains a state of } F \\ F & \text{otherwise} \end{cases}$$

and

$$\delta'(q, a) \stackrel{\text{def}}{=} \delta^*(q, a) \text{ for } q \in Q, \text{ and } a \in \Sigma.$$

Note: M' has no ϵ-moves

Example ∈-closure & Removal of ∈-moves



∈-closure of q_0 = $\{q_0, q_1, q_2, q_3\}$
∈-closure of q_1 = $\{q_1, q_3\}$
∈-closure of q_2 = $\{q_2, q_3\}$
∈-closure of q_3 = $\{q_3\}$
∈-closure of q_4 = $\{q_4\}$
∈-closure of q_5 = $\{q_5, q_3\}$

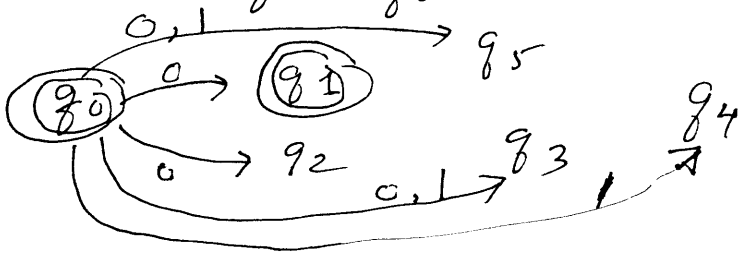
$\delta^*(q, a)$ = ∈-closure $(\delta(R, a))$ for $a \in \Sigma$
and $R = \delta^*(q, \lambda)$ = ∈-closure of q.

$\delta^*(q_0, a)$ = ∈-closure $(\delta($∈-closure of $q_0, a))$
= ∈-closure $(\delta(\{q_0, q_1, q_2, q_3\}, a))$

$\delta(\{q_0, q_1, q_2, q_3\}, 0) = \{q_1, q_3, q_5, q_2\}$
$\delta(\{q_0, q_1, q_2, q_3\}, 1) = \{q_5, q_3, q_4\}$

$\therefore$ $\delta^*(q_0, 0)$ = ∈-closure $\{q_1, q_3, q_5, q_2\} = \{q_1, q_2, q_3, q_5\}$
$\delta^*(q_0, 1)$ = ∈-closure $\{q_5, q_3, q_4\} = \{q_3, q_4, q_5\}$

$\therefore$ Transitions from q_0:



Similary for other states

Review Regular Grammars (Type 3 Grammars)

Given a Grammar G,

If each rule is of the form

$A \to xB$

or $A \to x$ where $x \in T^*$, $A, B \in N$

then

The grammar is ~~left~~ right linear.

If each rule is of the form

$A \to Bx$

or $A \to x$ where $x \in T^*$, $A, B, \in N$

then

The grammar is ~~right~~ left linear.

The grammar is regular if it is right linear or left linear.

Theorem

Every Language not containing λ that can be generated by a regular grammar as defined above can be generated by a right linear grammar with rules of the restricted form:

$A \to aB$

or $A \to a$ for $a \in T$ and $A, B \in N$.

(Similarly for left-linear)

Equivalence between Finite State Acceptors & Regular Grammars

Theorem

The set of finite state languages is exactly the set of languages generated by regular grammars.

Part 1.

L is a finite state language $\Rightarrow L = L(G)$ for some regular grammar G.

Proof: L f.s.l. $\Rightarrow$ there exists a d.f.s.a $M = \langle X, Q, \delta, q_0, F \rangle$ s.t. $T(M) = L$.

Note: $x \in T(M) \Leftrightarrow \tilde{\delta}(q_0, x) \in F$

Define the grammar G to be $G = \langle N, T, P, S \rangle$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\; \langle Q, X, P, q_0 \rangle$ (with $N = Q$, $T = X$, $S = q_0$)

with rules P:

for $q_i \xrightarrow{a} q_j$. have the rule

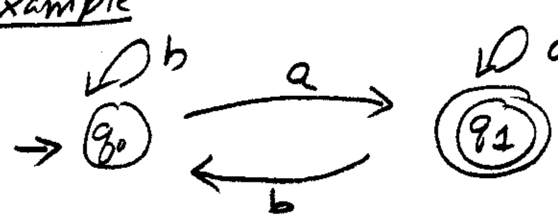
$$q_i \rightarrow a\, q_j$$

also, if $q_k \in F$ have the rule

$$q_k \rightarrow \lambda .$$

G is a right linear grammar that generates L.

Example



$F = \{q_1\}$

P:

$q_0 \to a q_1$ $\quad q_1 \to \lambda$.

$q_0 \to b q_0$

$q_1 \to a q_1$

$q_1 \to b q_0$

Prove that $T(M) = L(G)$.

Let $x = a_1 a_2 \dots a_n$. $\quad x \in T(M)$ $\therefore$

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} q_3 \to \dots \xrightarrow{a_n} q_n \in F.$$

By definition,

P contains rules $q_0 \to a_1 q_1$, $q_1 \to a_2 q_2$ $\dots$ $q_{n-1} \to a_n q_n$. and $q_n \to \lambda$.

$\therefore$

$$S \Rightarrow a_1 q_1 \Rightarrow a_1 a_2 q_2 \overset{*}{\Rightarrow} a_1 a_2 \dots a_n q_n \Rightarrow a_1 a_2 \dots a_n.$$

Alternatively,

if $x \in L(G)$ then,

$q_0 \Rightarrow a_1 q_1 \Rightarrow a_1 a_2 q_2 \Rightarrow \dots \Rightarrow a_1 \dots a_n q_n \Rightarrow a_1 \dots a_n$

But by definition, for each rule $\exists$

a transition $\delta(q_i, a_{i+1}) = q_{i+1}$

with $q_n \in F$ (since $q_n \to \lambda$ is a production)

$\therefore x = a_1 \dots a_n \in T(M)$.

Part 2.

$L = L(G)$ for a regular grammar $G \Rightarrow$ L is a f.s.l.

Proof:

Let $L = L(G)$ where $G = \langle N, T, P, S \rangle$ a regular grammar.

Construct the acceptor $M = (Q, X, \delta, q_0, F)$

where $Q = N \cup \{f\}$

$X = T$

$q_0 = S$

$F = \{f\}$

and δ is defined as follows:

For a production $A \to aB$ define the transition

$$\delta(A, a) = B$$

for a production of the form

$A \to a$ define the transition

$\delta(A, a) = f$ for $a \in \Sigma \cup \{\lambda\}$.

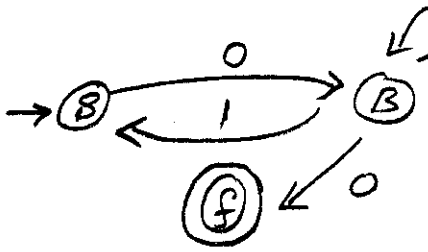
Example

$S \to 0B$

$B \to 0B$

$B \to 1S$

$B \to 0$



$S \Rightarrow 0B \Rightarrow 01S$

$\Rightarrow 010B \Rightarrow 0100B$

$\Rightarrow 01000$

$\in T(M)$

Note: this a non-deterministic f.s.a.

It can be easily shown that

$$x \in L(G) \iff x \in T(M)$$

Equivalence between Finite State Acceptors and Regular Sets

Theorem: (Kleene) L is a finite state language $\Leftrightarrow$ L is a regular set (ie. can be represented by a regular expression).

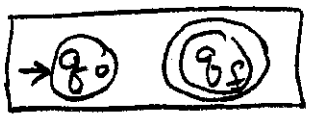
Part 1

Let r be a regular expression. Then there exists an n.f.a. that accepts r.

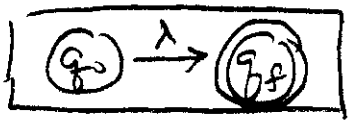
Proof outline.

Build an acceptor with 1 final state and no transitions out of it just as the regular expression is built up.

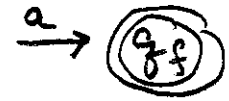
(1) to accept Φ



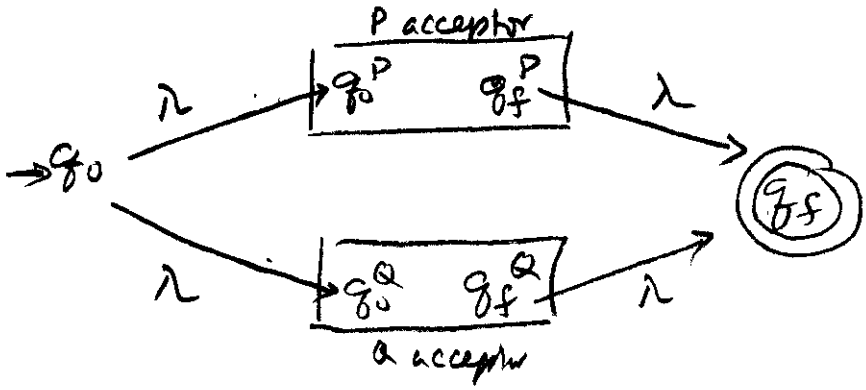
(2) to accept λ



(3) to accept any $a \in \Sigma$

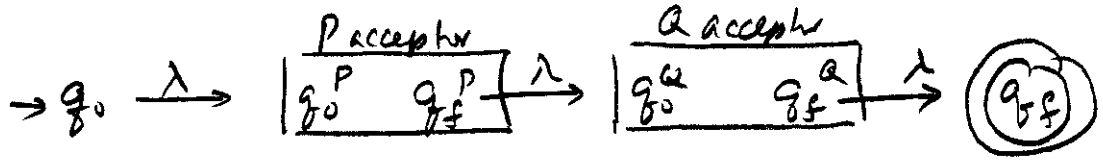


(4) to accept $P + Q$ where P, Q are regular expressions

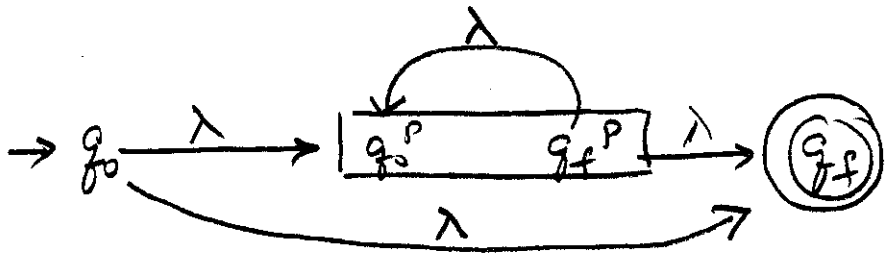


$ab + ac$

(5) to accept $P \cdot Q$

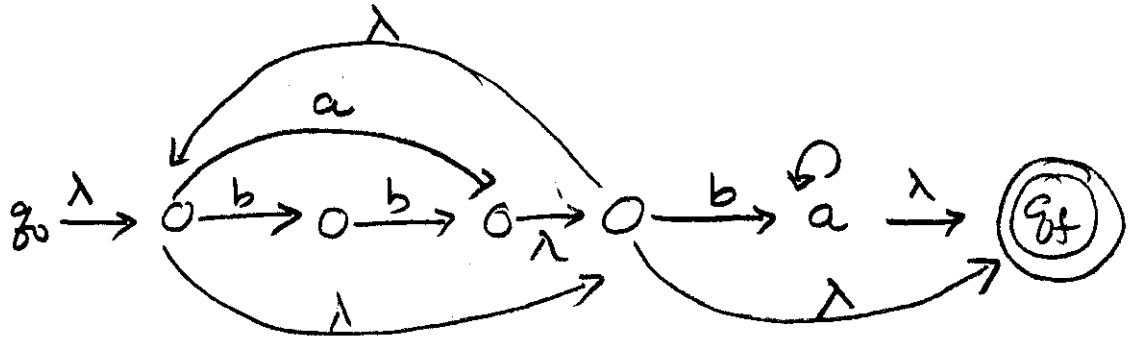


(6) to accept P^*



Can then prove that the combined machine M is such that $x \in T(M) \Leftrightarrow x \in L(r)$ for the regular expression r.

Example accepting $r = (a+bb)^*(ba^* + \lambda)$



Part 2

Let L be accepted by a d.f.a., ie $L = T(M)$ for some acceptor M. Then L can be denoted by a regular expression r such that $L = L(r)$.

Proof Let $L = T(M)$ for $M = (\{q_1, q_2, \ldots, q_n\}, \Sigma, \delta, q_1, F)$.
Let R_{ij}^k be the set of all strings x that take state q_i to q_j without any intermediate states higher than k.

Example R_{14}^3 $\quad q_1 \xrightarrow{a} q_3 \xrightarrow{b} q_2 \xrightarrow{a} q_4$

$aba \in R_{14}^3$ since q_1 goes to q_4 with no intermediate state higher than 3.

Note: R_{ij}^n is all strings that take one from q_i to q_j.

Now, $$R_{ij}^k = R_{ik}^{k-1} \left(R_{kk}^{k-1}\right)^* R_{kj}^{k-1} \cup R_{ij}^{k-1}$$

Also, $$R_{ij}^0 = \begin{cases} \{a \mid \delta(q_i, a) = q_j\}, & i \neq j \\ \{a \mid \delta(q_i, a) = q_j\} \cup \{\lambda\} & \text{for } i = j \end{cases}$$

We now show that for each i, j, k there exists a regular expression r_{ij}^k that denotes the set of strings R_{ij}^k. The proof is by induction on k

Basis $k=0$ Let r_{ij}^0 denote R_{ij}^0. r_{ij}^0 is a regular expression:
R_{ij}^0 = a finite set of strings of single symbols a_i or λ that go from q_i to q_j. $\therefore R_{ij}^0 = a_1 + a_2 + \cdots a_p$ (or ϕ)
where each a_i is a single symbol.

Induction $k-1 \to k$

Assume there exists a regular expression r_{lm}^{k-1} for any R_{lm}^{k-1} with $k \geq 1$.

We know that $R_{ij}^{k} = R_{ik}^{k-1} \left(R_{kk}^{k-1}\right)^{*} R_{kj}^{k-1} \cup R_{ij}^{k-1}$

By induction,
$\left(r_{ik}^{k-1}\right)\left(r_{kk}^{k-1}\right)^{*}\left(r_{kj}^{k-1}\right) + r_{ij}^{k-1}$ is a regular expression that denotes set R_{ij}^{k}.

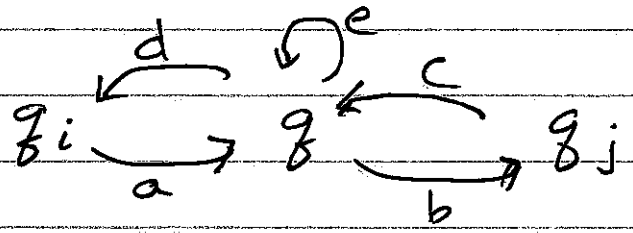
Finally, note that $L(M) = \bigcup_{q_j \in F} R_{1j}^{n}$

$\Rightarrow$ $L(M)$ is denoted by $\sum_{q_j \in F} r_{1j}$

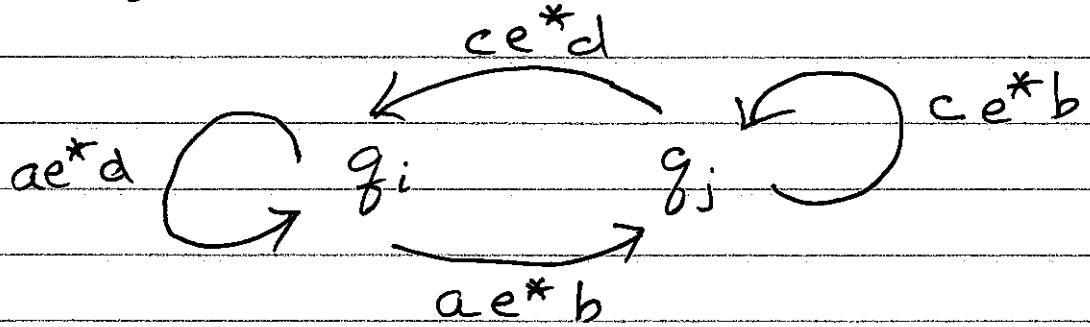
An alternative approach, discussed in the text uses to notion of a generalized transition graph.

Idea Extend labels on an nfa to be regular expressions. Given the nfa keep removing intermediate states until only the start state and unique final state remain.
(Note: can always turn any n.f.a into one that has a single final state)
Also, q_0 can be assumed not to be in F.

Example

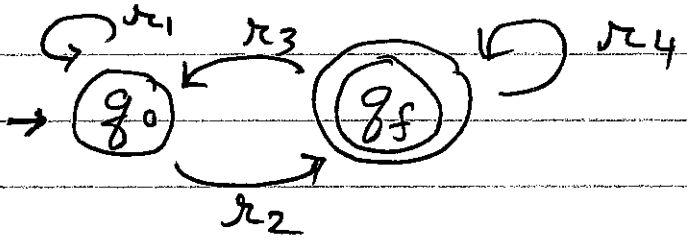


"Remove" q
to get:



Assume intermediate states have been removed until only start state & final state remain.

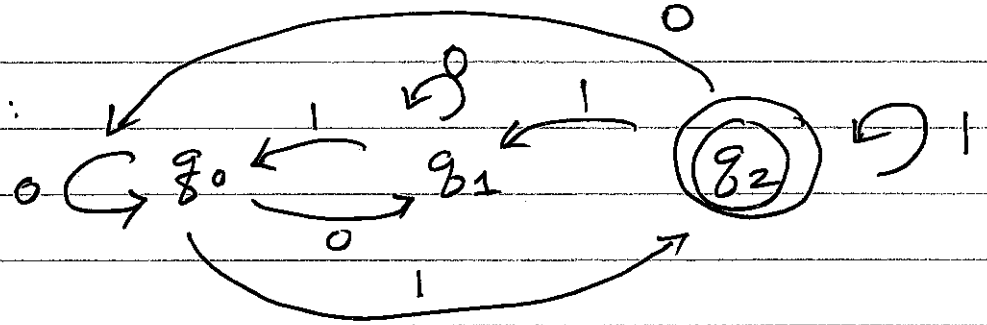
We would thus have:



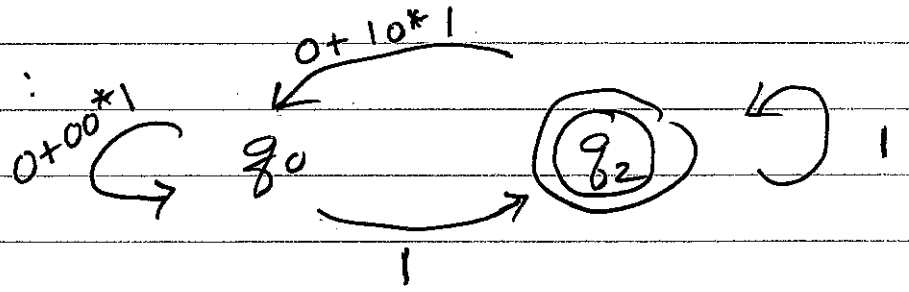
Claim: regular expression for this is simply

$$r = r_1^* r_2 (r_4 + r_3 r_1^* r_2)^*$$

Example:



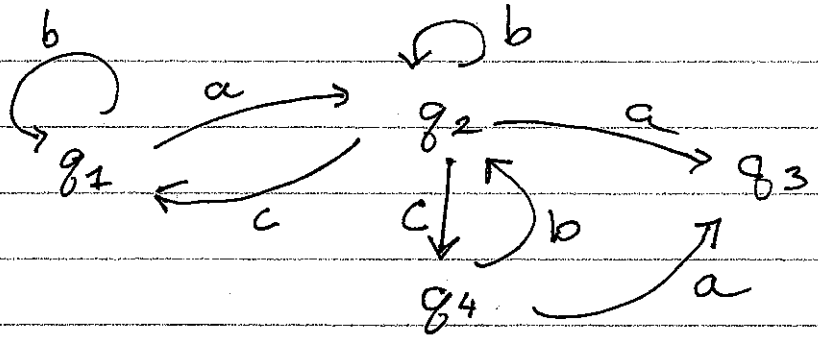
Eliminate q_1:



For this:

$$r = (0+00^*1)1(1 + (0+10^*1)(0+00^*1)^*1)^*$$

Example



"Elimination" of state q_2.

① Modify self loops on non q_2 vertices first

q_1: b goes to $b + ab^*c$

q_3: same ϕ (cannot go to q_2 & back)

q_4: $\phi \rightarrow bb^*c$

② Modify edge labels between inputs to q_2 and outputs from q_2

inputs : q_1, q_4 outputs : q_1, q_3, q_4

Consider pairs (q_1, q_3), (q_1, q_4), (q_4, q_1), (q_4, q_3)