Kleene's Theorem

Definition. A language is **regular** iff it is equal to $L(M)$, the set of strings accepted by some deterministic finite automaton M.

Theorem.

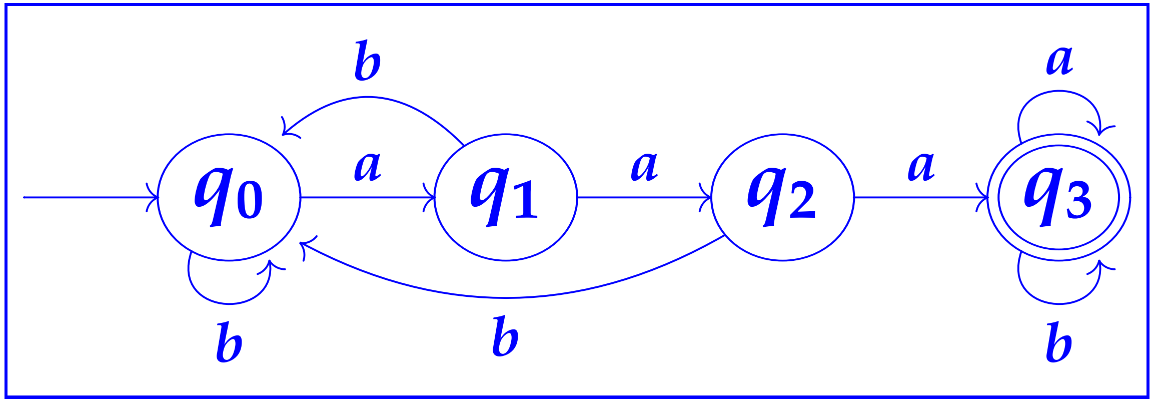
(a) For any regular expression r, the set $L(r)$ of strings matching r is a regular language.

(b) Conversely, every regular language is the form $L(r)$ for some regular expression r.

Example of a regular language

Recall the example DFA we used earlier:

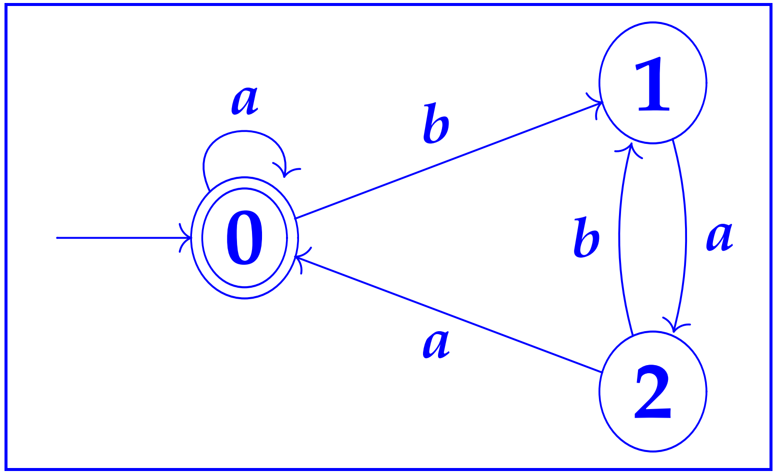
$M \triangleq$



In this case it's not hard to see that $L(M) = L(r)$ for

$$r = (a|b)^* aaa(a|b)^*$$

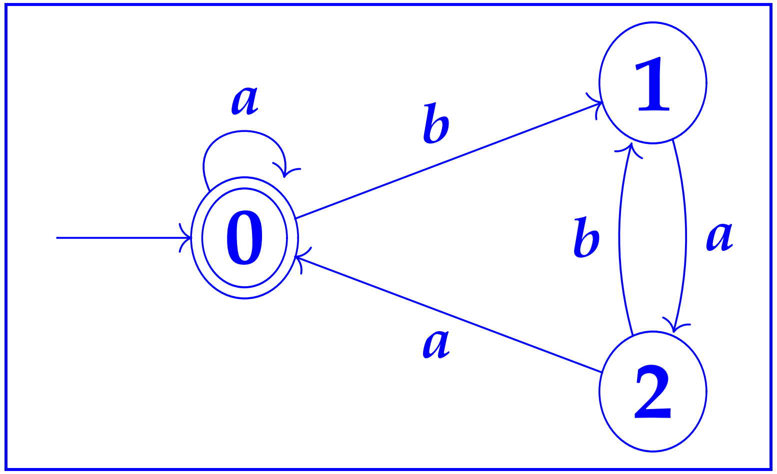
Example

$M \triangleq$ 

$L(M) = L(r)$ for which regular expression r?

Guess: $r = a^* | a^* b (ab)^* aaa^*$

Example

$M \triangleq$ 

$L(M) = L(r)$ for which regular expression r?

Guess: $r = a^*|a^*b(ab)^*aaa^*$

WRONG! since $baabaa \in L(M)$
but $baabaa \notin L(a^*|a^*b(ab)^*aaa^*)$

We need an algorithm for constructing a suitable r for each M (plus a proof that it is correct).

Lemma. Given an NFA $M = (Q, \Sigma, \Delta, s, F)$, for each subset $S \subseteq Q$ and each pair of states $q, q' \in Q$, there is a regular expression $r^S_{q,q'}$ satisfying

$$L(r^S_{q,q'}) = \{u \in \Sigma^* \mid q \xrightarrow{u}{}^* q' \text{ in } M \text{ with all intermediate states of the sequence of transitions in } S\}.$$

Hence if the subset F of accepting states has k distinct elements, $q_1, \ldots, q_k$ say, then $L(M) = L(r)$ with $r \triangleq r_1 | \cdots | r_k$ where

$$r_i = r^Q_{s,q_i} \qquad (i = 1, \ldots, k)$$

(in case $k = 0$, we take r to be the regular expression $\emptyset$).

Lemma on p83 is proved
by induction on # of elements in S

Base case $S = \emptyset$:

Given states q, q' in M, if

$$q \xrightarrow{a} q'$$

holds for just $a = a_1, \ldots, a_k$ then can take

$$r^{\emptyset}_{q,q'} \triangleq \begin{cases} a_1 | \cdots | a_k & \text{if } q \neq q' \\ a_1 | \cdots | a_k | \varepsilon & \text{if } q = q' \end{cases}$$

Lemma on p83 is proved
by induction on # of elements in S

Base case $S = \emptyset$:

Given states q, q' in M, if

$$q \xrightarrow{a} q'$$

holds for <u>no</u> a then can take

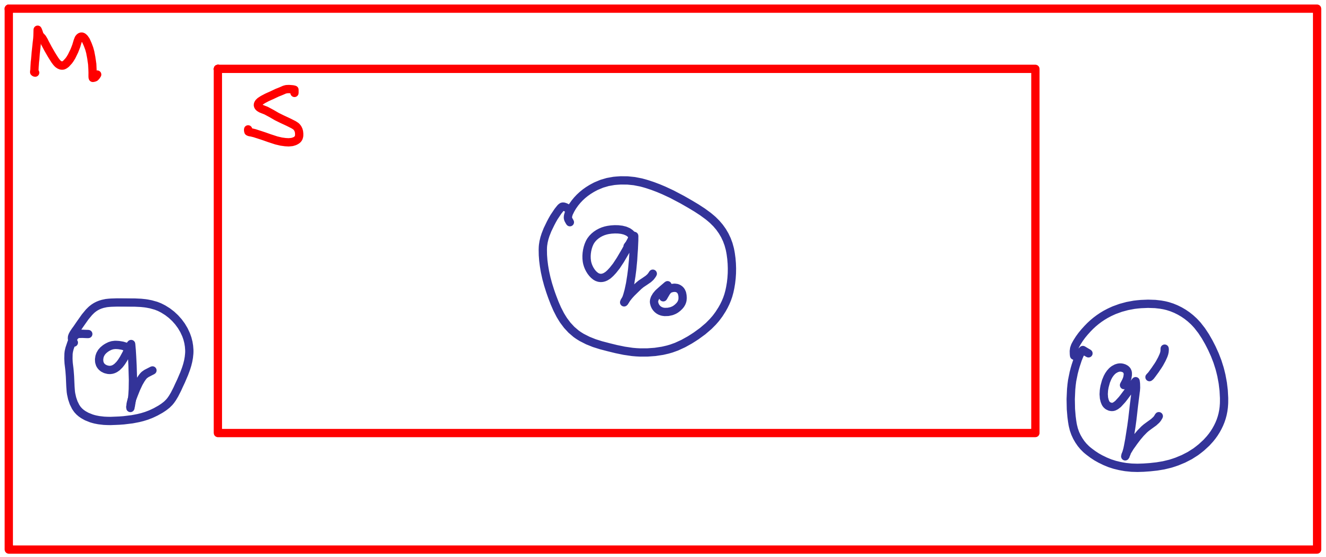
$$r^{\emptyset}_{q,q'} \triangleq \begin{cases} \emptyset & \text{if } q \neq q' \\ \varepsilon & \text{if } q = q' \end{cases}$$

Induction step: S has $n+1$ elements

Pick any $q_0 \in S$. So can apply induction hyp. to $S \setminus \{q_0\} = \{q \in S \mid q \neq q_0\}$ since it has n elts.

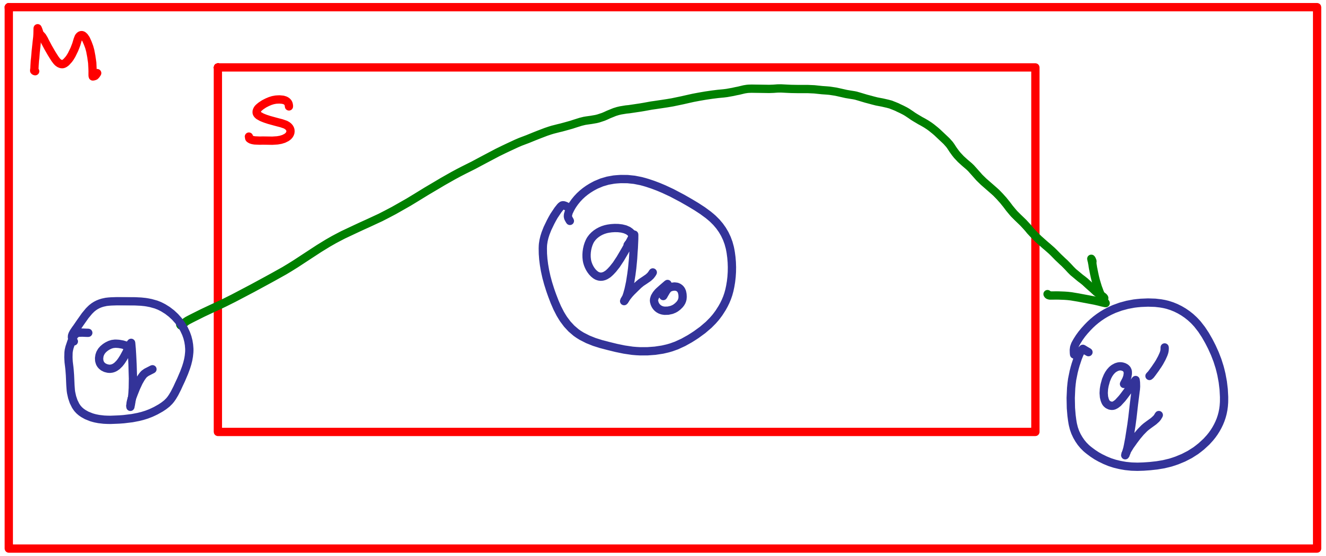
Induction step: S has $n+1$ elements

Pick any $q_0 \in S$. So can apply induction hyp. to $S \setminus \{q_0\} = \{q \in S \mid q \neq q_0\}$ since it has n elts.



Induction step: S has $n+1$ elements

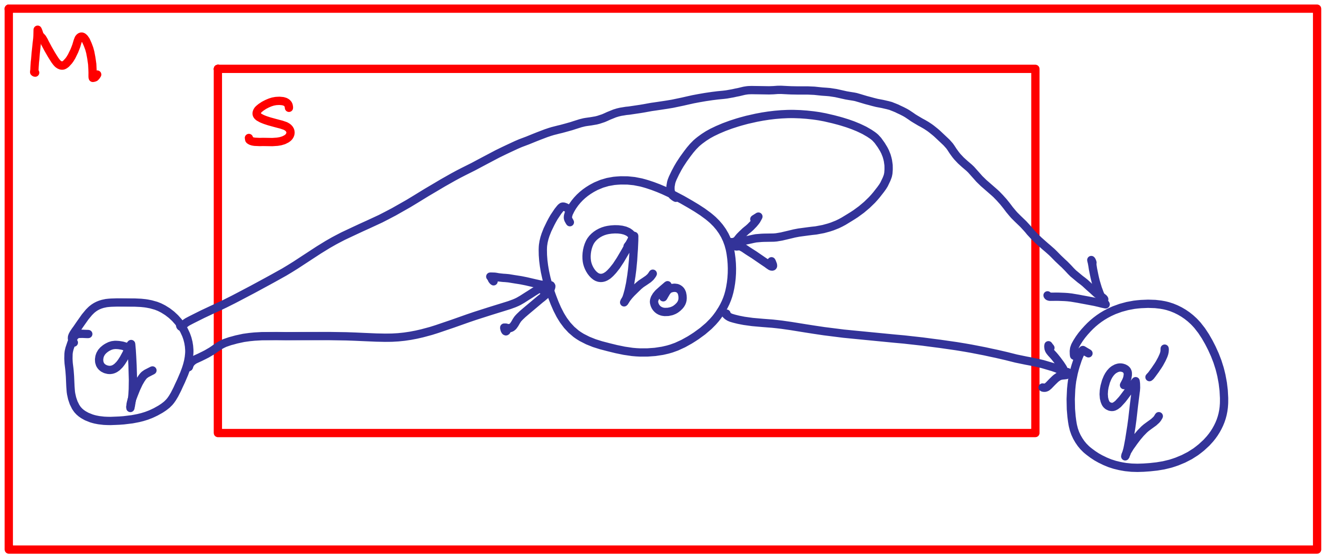
Pick any $q_0 \in S$. So can apply induction hyp. to $S \setminus \{q_0\} = \{q \in S \mid q \neq q_0\}$ since it has n elts.



$$r^{S}_{q,q'} = r^{S\setminus\{q_0\}}_{q,q'} \ \ldots$$

Induction step: S has $n+1$ elements

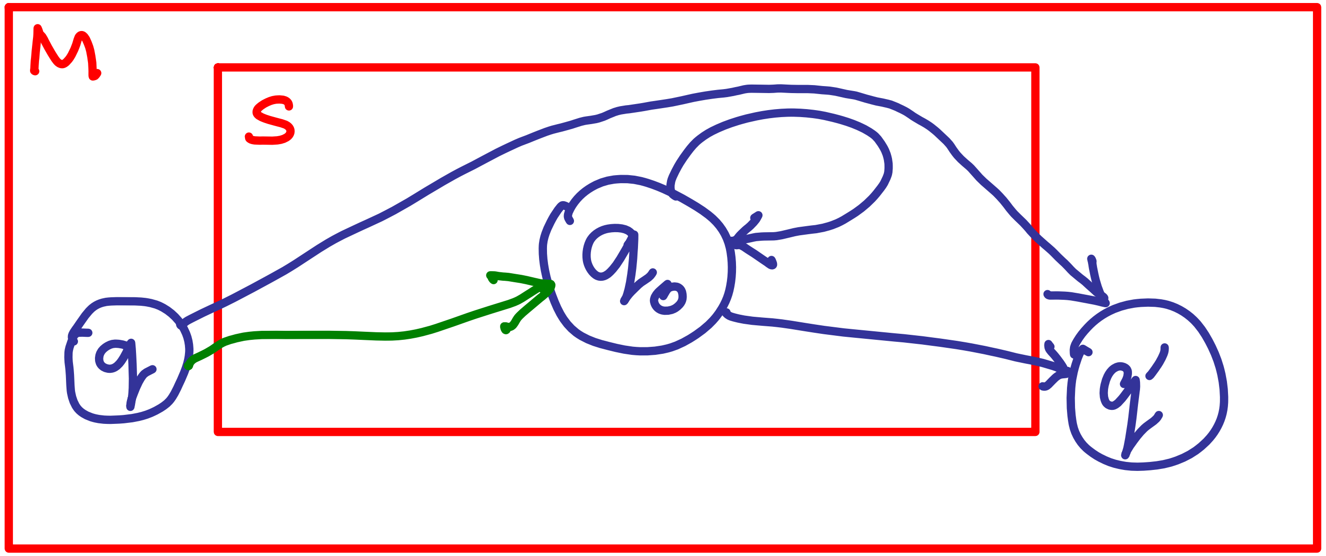
Pick any $q_0 \in S$. So can apply induction hyp. to $S \setminus \{q_0\} = \{q \in S \mid q \neq q_0\}$ since it has n elts.



$$r^{S}_{q,q'} = r^{S\setminus\{q_0\}}_{q,q'} \mid r^{S\setminus\{q_0\}}_{q,q_0} \left(r^{S\setminus\{q_0\}}_{q_0,q_0}\right)^{*} r^{S\setminus\{q_0\}}_{q_0,q'}$$

Induction step: S has $n+1$ elements

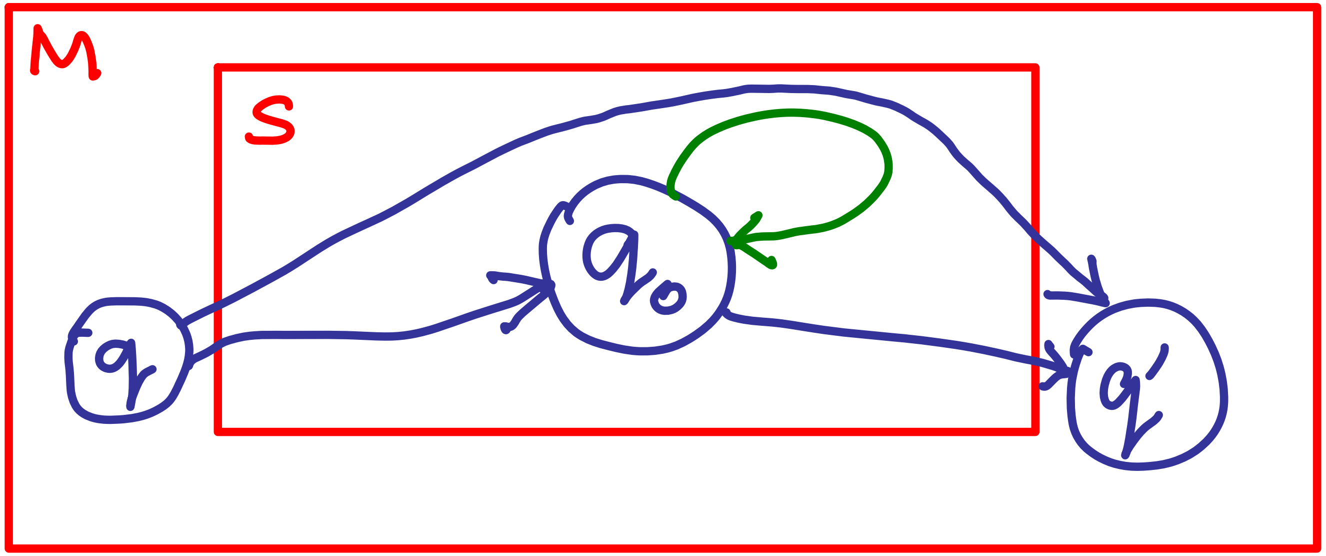
Pick any $q_0 \in S$. So can apply induction hyp. to $S \setminus \{q_0\} = \{q \in S \mid q \neq q_0\}$ since it has n elts.



$$r^S_{q,q'} = r^{S\setminus\{q_0\}}_{q,q'} \mid r^{S\setminus\{q_0\}}_{q,q_0} \left(r^{S\setminus\{q_0\}}_{q_0,q_0}\right)^* r^{S\setminus\{q_0\}}_{q_0,q'}$$

Induction step: S has $n+1$ elements

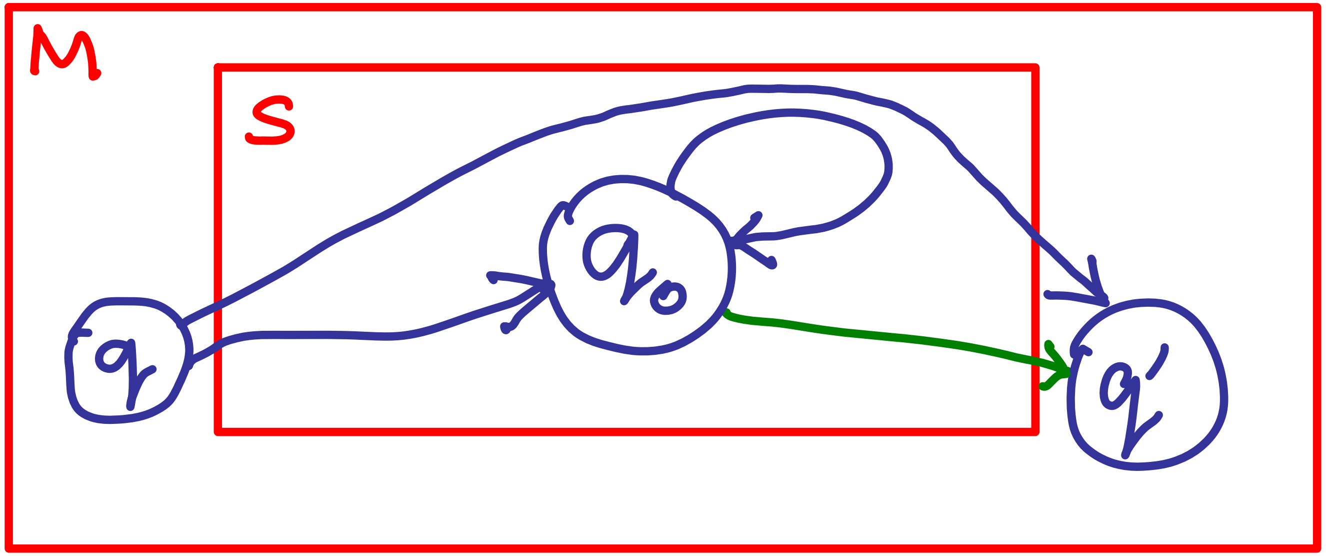
Pick any $q_0 \in S$. So can apply induction hyp. to $S\setminus\{q_0\} = \{q \in S \mid q \neq q_0\}$ since it has n elts.



$$r^{S}_{q,q'} = r^{S\setminus\{q_0\}}_{q,q'} \mid r^{S\setminus\{q_0\}}_{q,q_0} \left(r^{S\setminus\{q_0\}}_{q_0,q_0}\right)^* r^{S\setminus\{q_0\}}_{q_0,q'}$$

Induction step: S has $n+1$ elements

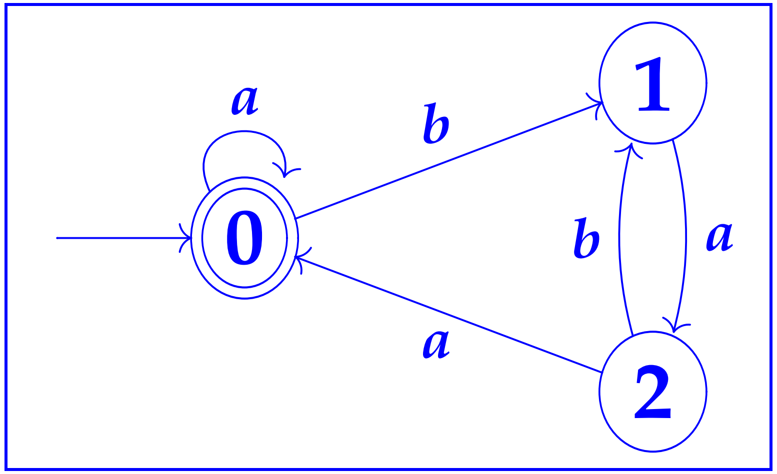
Pick any $q_0 \in S$. So can apply induction hyp. to $S \setminus \{q_0\} = \{q \in S \mid q \neq q_0\}$ since it has n elts.



$$r^{S}_{q,q'} = r^{S\setminus\{q_0\}}_{q,q'} \mid r^{S\setminus\{q_0\}}_{q,q_0}\left(r^{S\setminus\{q_0\}}_{q_0,q_0}\right)^{*} r^{S\setminus\{q_0\}}_{q_0,q'}$$

$r^{\{0,1,2\}}_{0,0}$

$M \triangleq$



By direct inspection we have:

$r^{\{0\}}_{i,j}$	0	1	2
0			
1	$\emptyset$	ε	a
2	aa^*	a^*b	ε

$r^{\{0,2\}}_{i,j}$	0	1	2
0	a^*	a^*b	
1			
2			

(we don't need the unfilled entries in the tables)

Example p 87

Want $r_{0,0}^{\{0,1,2\}}$

Remove 1 from $\{0,1,2\}$

$$r_{0,0}^{\{0,1,2\}} \triangleq \underbrace{r_{0,0}^{\{0,2\}}}_{a^*} \mid \underbrace{r_{0,1}^{\{0,2\}}}_{a^*b} \left(r_{1,1}^{\{0,2\}}\right)^* r_{1,0}^{\{0,2\}}$$

Example p 87 Want $r_{0,0}^{\{0,1,2\}}$

$$r_{0,0}^{\{0,1,2\}} = a^* \mid a^* b \left(r_{1,1}^{\{0,2\}}\right)^* r_{1,0}^{\{0,1\}}$$

$$r_{1,1}^{\{0,2\}} \triangleq r_{1,1}^{\{0\}} \mid r_{1,2}^{\{0\}} \left(r_{2,2}^{\{0\}}\right)^* r_{2,1}^{\{0\}}$$

$$= \varepsilon \mid a \; (\varepsilon)^* \; a^* b$$

$$= \varepsilon \mid a a^* b$$

equivalence: $r = s \triangleq L(r) = L(s)$

Example p 87

Want $r_{0,0}^{\{0,1,2\}}$

$$r_{0,0}^{\{0,1,2\}} = a^* \mid a^*b(\varepsilon \mid aa^*b)^* r_{1,0}^{\{0,1\}}$$

$$\begin{aligned} r_{1,0}^{\{0,1\}} &\triangleq r_{1,0}^{\{0\}} \mid r_{1,2}^{\{0\}} (r_{2,2}^{\{0\}})^* r_{2,0}^{\{0\}} \\ &= \emptyset \mid a \ (\varepsilon)^* \ aa^* \\ &= aaa^* \end{aligned}$$

Example p 87 Want $r_{0,0}^{\{0,1,2\}}$

$$r_{0,0}^{\{0,1,2\}} = a^* \mid a^*b(\varepsilon \mid aa^*b)^* aaa^*$$

Some questions

(a) Is there an algorithm which, given a string u and a regular expression r, computes whether or not u matches r?

(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions r and s, computes whether or not they are **equivalent**, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of Σ^*) of the form $L(r)$ for some r?

$Not(M)$

Given DFA $M = (Q, \Sigma, \delta, s, F)$,
then $Not(M)$ is the DFA with

- set of states $= Q$
- input alphabet $= \Sigma$
- next-state function $= \delta$
- start state $= s$
- accepting states $= \{q \in Q \mid q \notin F\}$.

(i.e. we just reverse the role of accepting/non-accepting and leave everything else the same)

Because M is a *deterministic* finite automaton, then u is accepted by $Not(M)$ iff it is not accepted by M:

$$L(Not(M)) = \{u \in \Sigma^* \mid u \notin L(M)\}$$

[p 90]

Given reg. exp. r

Can construct reg. exp. $\sim r$

such that $\boxed{L(\sim r) = \{u \in \Sigma^* \mid u \notin L(r)\}}$

[p90]

Given reg. exp. r

Can construct reg. exp. $\sim r$

such that $\boxed{L(\sim r) = \{u \in \Sigma^* \mid u \notin L(r)\}}$

$$r \xrightarrow{\text{Kleene (a)}} M$$

$L(M) = L(r)$

[p90]

Given reg. exp. r

Can construct reg. exp. $\sim r$

such that $\boxed{L(\sim r) = \{u \in \Sigma^* \mid u \notin L(r)\}}$

$$r \xrightarrow{\text{Kleene (a)}} M \qquad Not(M) \xrightarrow{\text{Kleene (b)}} \sim r$$

$L(M) = L(r)$ $\qquad$ $L(\sim r) = L(Not(M))$

[p90]

Given reg. exp. r

Can construct reg. exp. $\sim r$

such that $\boxed{L(\sim r) = \{u \in \Sigma^* \mid u \notin L(r)\}}$

$$r \xrightarrow{\text{Kleene (a)}} M \qquad Not(M) \xrightarrow{\text{Kleene (b)}} \sim r$$

$$L(M) = L(r) \qquad L(\sim r) = L(Not(M))$$

so: $L(\sim r) = L(Not(M)) = \Sigma^* \setminus L(M) = \Sigma^* \setminus L(r)$

Regular languages are closed under intersection

Theorem. If L_1 and L_2 are a regular languages over an alphabet Σ, then their intersection $L_1 \cap L_2 = \{u \in \Sigma^* \mid u \in L_1 \,\&\, u \in L_2\}$ is also regular.

Proof. Note that $L_1 \cap L_2 = \Sigma^* \setminus ((\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2))$

(*cf.* de Morgan's Law: $p \,\&\, q = \neg(\neg p \vee \neg q)$).

So if $L_1 = L(M_1)$ and $L_2 = L(M_2)$ for DFAs M_1 and M_2, then $L_1 \cap L_2 = L(Not(PM))$ where M is the NFA$^\varepsilon$ $Union(Not(M_1), Not(M_2))$. $\square$

[It is not hard to directly construct a DFA $And(M_1, M_2)$ from M_1 and M_2 such that $L(And(M_1, M_2)) = L(M_1) \cap L(M_2)$ – see Exercise 4.7.]

Regular languages are closed under intersection

Corollary: given regular expressions r_1 and r_2,there is a regular expression, which we write as $r_1 \,\&\, r_2$, such that

> a string u matches $r_1 \,\&\, r_2$ iff it matches both r_1 and r_2.

Proof. By Kleene (a), $L(r_1)$ and $L(r_2)$ are regular languages and hence by the theorem, so is $L(r_1) \cap L(r_2)$. Then we can use Kleene (b) to construct a regular expression $r_1 \,\&\, r_2$ with $L(r_1 \,\&\, r_2) = L(r_1) \cap L(r_2)$. □