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.

The first part requires us to demonstrate that for any regular expression r, we can construct a DFA, M with $L(M) = L(r)$

We will do this by demonstrating that for any r we can construct a NFA$^{\varepsilon}$ M' with $L(M') = L(r)$ and rely on the subset construction theorem to give us the DFA M.

We consider each axiom and rule that define regular expressions

Axioms

$U = (\Sigma \cup \Sigma')^*$

axioms: $\dfrac{}{a}$ $\dfrac{}{\epsilon}$ $\dfrac{}{\emptyset}$

(where $a \in \Sigma$ and $r, s \in U$)

with straightforward matching rules

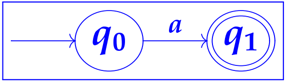
Axioms

$$U = (\Sigma \cup \Sigma')^*$$

axioms: $\dfrac{\quad}{a}$ $\qquad$ $\dfrac{\quad}{\epsilon}$ $\qquad$ $\dfrac{\quad}{\emptyset}$

(where $a \in \Sigma$ and $r, s \in U$)

with straightforward matching rules



just accepts the one-symbol string a

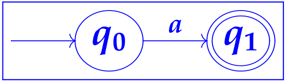
Axioms

$U = (\Sigma \cup \Sigma')^*$

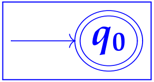
axioms: $\dfrac{}{a}$ $\dfrac{}{\epsilon}$ $\dfrac{}{\emptyset}$

(where $a \in \Sigma$ and $r, s \in U$)

with straightforward matching rules



just accepts the one-symbol string a



just accepts the null string, ε

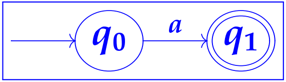
Axioms

$U = (\Sigma \cup \Sigma')^*$

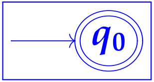
axioms: $\dfrac{}{a}$ $\qquad \dfrac{}{\epsilon}$ $\qquad \dfrac{}{\emptyset}$

(where $a \in \Sigma$ and $r, s \in U$)

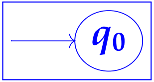
with straightforward matching rules



just accepts the one-symbol string a



just accepts the null string, ε



accepts no strings

Kleene's Theorem Part a (The Fun Part)

For any regular expression r we can build an NFA$^{\varepsilon}$ M such that $L(r) = L(M)$

We will work on induction on the depth of abstract syntax trees

Recall: Regular expressions (abstract syntax)

The ‘signature’ for regular expression abstract syntax trees (over an alphabet Σ) consists of

- binary operators Union and Concat
- unary operator Star
- nullary operators (constants) Null, Empty and Sym_a (one for each $a \in \Sigma$).

Recall: Regular expressions (abstract syntax)

(concrete syntax)

The ‘signature’ for regular expression abstract syntax trees (over an alphabet Σ) consists of

- binary operators Union and Concat
- unary operator Star
- nullary operators (constants) Null, Empty and Sym_a (one for each $a \in \Sigma$).

Recall: Regular expressions (abstract syntax)

(concrete syntax)

The ‘signature’ for regular expression abstract syntax trees (over an alphabet Σ) consists of

- binary operators Union and Concat

  $r_1|r_2$ $\quad$ r_1r_2

- unary operator Star
- nullary operators (constants) Null, Empty and Sym_a (one for each $a \in \Sigma$).

Recall: Regular expressions (abstract syntax)

(concrete syntax)

The 'signature' for regular expression abstract syntax trees (over an alphabet Σ) consists of

- binary operators *Union* and *Concat*
  $r_1 | r_2$ $r_1 r_2$
- unary operator *Star* r^*
- nullary operators (constants) *Null*, *Empty* and Sym_a (one for each $a \in \Sigma$).

Recall: Regular expressions (abstract syntax)

(concrete syntax)

The 'signature' for regular expression abstract syntax trees (over an alphabet Σ) consists of

- binary operators $Union$ and $Concat$
  $r_1|r_2$ $r_1 r_2$
- unary operator $Star$ r^*
- nullary operators (constants) $Null$, $Empty$ and Sym_a (one for each $a \in \Sigma$).
  ϵ $\emptyset$ a

(i) **Base cases:** show that $\{a\}$, $\{\varepsilon\}$ and $\emptyset$ are regular languages.

(ii) **Induction step for $r_1|r_2$:** given NFA$^\varepsilon$s M_1 and M_2, construct an NFA$^\varepsilon$ $Union(M_1, M_2)$ satisfying

$$\boxed{L(Union(M_1, M_2)) = \{u \mid u \in L(M_1) \vee u \in L(M_2)\}}$$

Thus if $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$, then $L(r_1|r_2) = L(Union(M_1, M_2))$.

(i) **Base cases:** show that $\{a\}$, $\{\varepsilon\}$ and $\emptyset$ are regular languages.

(ii) **Induction step for $r_1|r_2$:** given NFA$^{\varepsilon}$s M_1 and M_2, construct an NFA$^{\varepsilon}$ $Union(M_1, M_2)$ satisfying

$$L(Union(M_1, M_2)) = \{u \mid u \in L(M_1) \vee u \in L(M_2)\}$$

Thus if $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$, then $L(r_1|r_2) = L(Union(M_1, M_2))$.

(iii) **Induction step for r_1r_2:** given NFA$^{\varepsilon}$s M_1 and M_2, construct an NFA$^{\varepsilon}$ $Concat(M_1, M_2)$ satisfying

$$L(Concat(M_1, M_2)) = \{u_1u_2 \mid u_1 \in L(M_1) \ \& \ u_2 \in L(M_2)\}$$

Thus $L(r_1r_2) = L(Concat(M_1, M_2))$ when $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$.

(i) **Base cases:** show that $\{a\}$, $\{\varepsilon\}$ and $\emptyset$ are regular languages.

(ii) **Induction step for $r_1|r_2$:** given NFA$^\varepsilon$s M_1 and M_2, construct an NFA$^\varepsilon$ $Union(M_1, M_2)$ satisfying

$$L(Union(M_1, M_2)) = \{u \mid u \in L(M_1) \vee u \in L(M_2)\}$$

Thus if $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$, then $L(r_1|r_2) = L(Union(M_1, M_2))$.

(iii) **Induction step for r_1r_2:** given NFA$^\varepsilon$s M_1 and M_2, construct an NFA$^\varepsilon$ $Concat(M_1, M_2)$ satisfying

$$L(Concat(M_1, M_2)) = \{u_1u_2 \mid u_1 \in L(M_1) \,\&\, u_2 \in L(M_2)\}$$

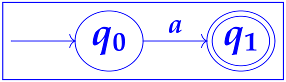
Thus $L(r_1r_2) = L(Concat(M_1, M_2))$ when $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$.

(iv) **Induction step for r^*:** given NFA$^\varepsilon$ M, construct an NFA$^\varepsilon$ $Star(M)$ satisfying

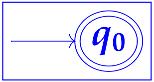
$$L(Star(M)) = \{u_1u_2 \ldots u_n \mid n \geq 0 \text{ and each } u_i \in L(M)\}$$

Thus $L(r^*) = L(Star(M))$ when $L(r) = L(M)$.

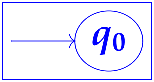
NFAs for regular expressions a, ϵ, $\emptyset$



just accepts the one-symbol string a

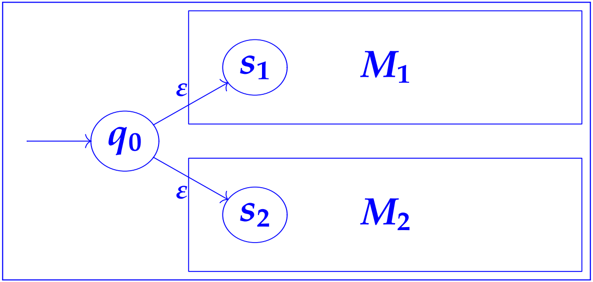


just accepts the null string, ε



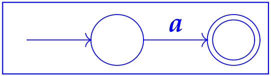
accepts no strings

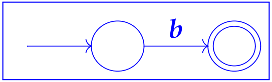
$Union(M_1, M_2)$

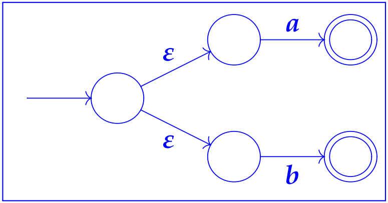


accepting states = union of accepting states of M_1 and M_2

For example,

if $M_a =$ 

and $M_b =$ 

then $Union(M_a, M_b) =$ 

In what follows, whenever we have to deal with two machines, say M_1 and M_2 together, we assume that their states are disjoint.

If they were not, we could just rename the states of one machine to make this so.

Also assume that for r_1 and r_2 there are machines M_1 and M_2 such that $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$

Construction for $Union(r_1, r_2)$

Assume there are two machines M_1 and M_2 with $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$

Construction for $Union(r_1, r_2)$

Assume there are two machines M_1 and M_2 with $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$

States of new machine $M = Union(M_1, M_2)$ are all the states in M_1 and all the states in M_2 together with a new start state with ε-transitions to each of the (old) start states of M_1 and M_2.

Construction for $Union(r_1, r_2)$

Assume there are two machines M_1 and M_2 with $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$

States of new machine $M = Union(M_1, M_2)$ are all the states in M_1 and all the states in M_2 together with a new start state with ε-transitions to each of the (old) start states of M_1 and M_2.

Accept states of M are the all accept states in M_1 and all accept states in M_2.

Construction for $Union(r_1, r_2)$

Assume there are two machines M_1 and M_2 with $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$

States of new machine $M = Union(M_1, M_2)$ are all the states in M_1 and all the states in M_2 together with a new start state with ε-transitions to each of the (old) start states of M_1 and M_2.

Accept states of M are the all accept states in M_1 and all accept states in M_2.

The transitions of M are all transitions in M_1 and M_2 along with the two ε-transitions from the new start state

M accepts any strings that M_1 accepts:

if $u \in L(M_1)$ then $s_1 \overset{u}{\Rightarrow} q_1$ where s_1 is start state and q_1 an accept state of M_1 respectively.

M accepts any strings that M_1 accepts:

if $u \in L(M_1)$ then $s_1 \stackrel{u}{\Rightarrow} q_1$ where s_1 is start state and q_1 an accept state of M_1 respectively.

But then in M, $s \stackrel{u}{\Rightarrow} q_1$, where s is our new start state since $s \xrightarrow{\varepsilon} s_1$.

M accepts any strings that M_1 accepts:

if $u \in L(M_1)$ then $s_1 \stackrel{u}{\Rightarrow} q_1$ where s_1 is start state and q_1 an accept state of M_1 respectively.

But then in M, $s \stackrel{u}{\Rightarrow} q_1$, where s is our new start state since $s \xrightarrow{\varepsilon} s_1$.

so $u \in L(M)$. Similar argument for M accepting any string that M_2 accepts

M accepts any strings that M_1 accepts:

if $u \in L(M_1)$ then $s_1 \stackrel{u}{\Rightarrow} q_1$ where s_1 is start state and q_1 an accept state of M_1 respectively.

But then in M, $s \stackrel{u}{\Rightarrow} q_1$, where s is our new start state since $s \xrightarrow{\varepsilon} s_1$.

so $u \in L(M)$. Similar argument for M accepting any string that M_2 accepts

so $(L(M_1) \cup L(M_2)) \subseteq L(Union(M_1, M_2))$

Can M accept anything more?

Can M accept anything more?

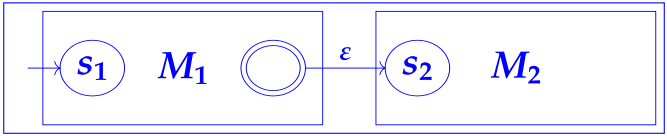
The only way "out of" s, the start state of M, is either to the start state of M_1 or the start state of M_2

Can M accept anything more?

The only way "out of" s, the start state of M, is either to the start state of M_1 or the start state of M_2

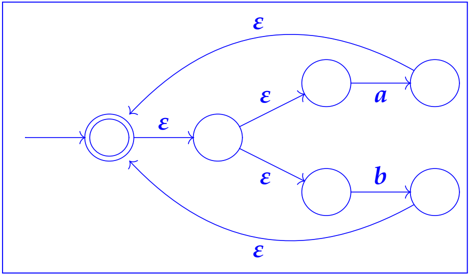
So no, $L(M) = (L(M_1) \cup L(M_2))$

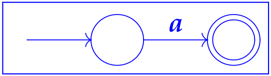
$Concat(M_1, M_2)$

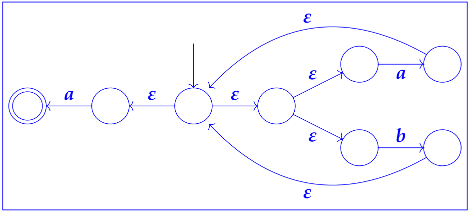


accepting states are those of M_2

For example,

if $M_1 =$ 

and $M_2 =$ 

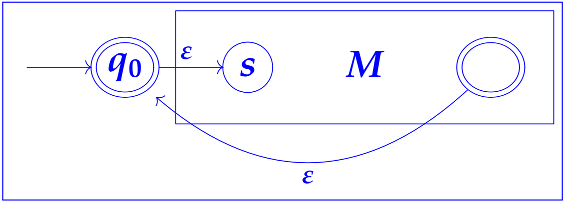
then $Concat(M_1, M_2) =$ 

Construction for $M = Concat(M_1, M_2)$

Make an ε-transition from every accept state in M_1 to the start state of M_2.

Start state of M is the start state of M_1; accept states of M are the accept states of M_2

$Star(M)$

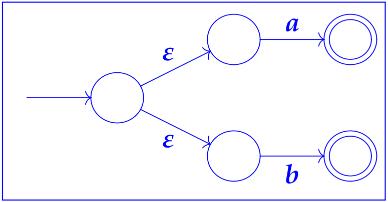


the only accepting state of $Star(M)$ is q_0

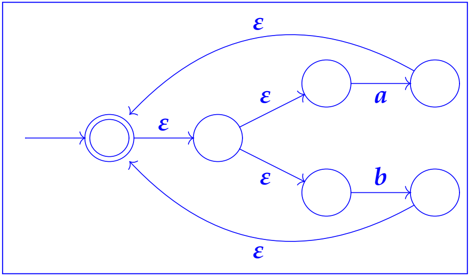
(N.B. doing without q_0 by just looping back to s
and making that accepting won't work – Exercise 4.1.)

For example,

if $M =$



then $Star(M) =$



Construction for $Star(r_1)$, $M = Star(M_1)$

Create a new state, say s which will be the start state, and the only accepting state of M.

Construction for $Star(r_1)$, $M = Star(M_1)$

Create a new state, say s which will be the start state, and the only accepting state of M.

The transitions of M are all the transitions of M_1 together with an ε-transition from s to the (old) start state of M_1 and ε-transitions from every (old) accepting state of M_1 to s.

Construction for $Star(r_1)$, $M = Star(M_1)$

Create a new state, say s which will be the start state, and the only accepting state of M.

The transitions of M are all the transitions of M_1 together with an ε-transition from s to the (old) start state of M_1 and ε-transitions from every (old) accepting state of M_1 to s.

Clearly, M accepts ε since s, the start state, is also an accepting state

Construction for $Star(r_1)$, $M = Star(M_1)$

Create a new state, say s which will be the start state, and the only accepting state of M.

The transitions of M are all the transitions of M_1 together with an ε-transition from s to the (old) start state of M_1 and ε-transitions from every (old) accepting state of M_1 to s.

Clearly, M accepts ε since s, the start state, is also an accepting state

nonempty strings accepted by M have to be formed of components, each of which is accepted by M_1

Construction for $Star(r_1)$, $M = Star(M_1)$

Create a new state, say s which will be the start state, and the only accepting state of M.

The transitions of M are all the transitions of M_1 together with an ε-transition from s to the (old) start state of M_1 and ε-transitions from every (old) accepting state of M_1 to s.

Clearly, M accepts ε since s, the start state, is also an accepting state

nonempty strings accepted by M have to be formed of components, each of which is accepted by M_1

so $L(M) = L(r_1^*)$

(i) **Base cases:** show that $\{a\}$, $\{\varepsilon\}$ and $\emptyset$ are regular languages.

(ii) **Induction step for $r_1|r_2$:** given NFA$^\varepsilon$s M_1 and M_2, construct an NFA$^\varepsilon$ $Union(M_1, M_2)$ satisfying

$$L(Union(M_1, M_2)) = \{u \mid u \in L(M_1) \lor u \in L(M_2)\}$$

Thus if $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$, then $L(r_1|r_2) = L(Union(M_1, M_2))$.

(iii) **Induction step for r_1r_2:** given NFA$^\varepsilon$s M_1 and M_2, construct an NFA$^\varepsilon$ $Concat(M_1, M_2)$ satisfying

$$L(Concat(M_1, M_2)) = \{u_1u_2 \mid u_1 \in L(M_1) \ \& \ u_2 \in L(M_2)\}$$

Thus $L(r_1r_2) = L(Concat(M_1, M_2))$ when $L(r_1) = L(M_1)$ and $L(r_2) = L(M_2)$.

(iv) **Induction step for r^*:** given NFA$^\varepsilon$ M, construct an NFA$^\varepsilon$ $Star(M)$ satisfying

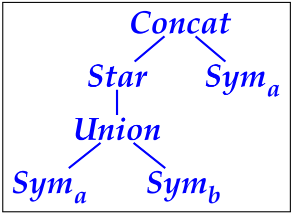
$$L(Star(M)) = \{u_1u_2 \dots u_n \mid n \geq 0 \text{ and each } u_i \in L(M)\}$$

Thus $L(r^*) = L(Star(M))$ when $L(r) = L(M)$.

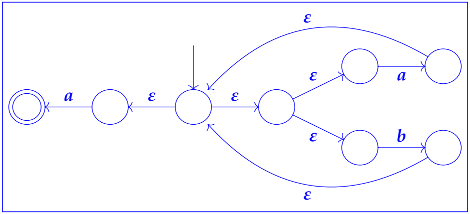
Example

Regular expression $(a|b)^*a$

whose abstract syntax tree is



is mapped to the NFA$^\varepsilon$ $Concat(Star(Union(M_a, M_b)), M_a) =$



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?

Decidability of matching

We now have a positive answer to question (a). Given string u and regular expression r:

- construct an NFA$^\varepsilon$ M satisfying $L(M) = L(r)$;
- in PM (the DFA obtained by the subset construction) carry out the sequence of transitions corresponding to u from the start state to some state q (because PM is deterministic, there is a unique such transition sequence);
- check whether q is accepting or not: if it is, then $u \in L(PM) = L(M) = L(r)$, so u matches r; otherwise $u \notin L(PM) = L(M) = L(r)$, so u does not match r.

(The subset construction produces an exponential blow-up of the number of states: PM has 2^n states if M has n. This makes the method described above potentially inefficient – more efficient algorithms exist that don't construct the whole of PM.)

Exponential Blow-up

if NFA^{ε} M has n states then the DFA made by subset construction, PM has 2^n states, since its states are the members of the powerset of M.

Minimisation of states in PM by:

Exponential Blow-up

if NFA^{ε} M has n states then the DFA made by subset construction, PM has 2^n states, since its states are the members of the powerset of M.

Minimisation of states in PM by:

- removing all states which are not reachable (by any string) from the start state.

Exponential Blow-up

if NFA^{ε} M has n states then the DFA made by subset construction, PM has 2^n states, since its states are the members of the powerset of M.

Minimisation of states in PM by:

- removing all states which are not reachable (by any string) from the start state.
- merge all compatible states. Two states are compatible if (i) they are both accepting or both non-accepting; and (ii) their transition functions are the same.

Exponential Blow-up

if NFA^ε M has n states then the DFA made by subset construction, PM has 2^n states, since its states are the members of the powerset of M.

Minimisation of states in PM by:

- removing all states which are not reachable (by any string) from the start state.
- merge all compatible states. Two states are compatible if (i) they are both accepting or both non-accepting; and (ii) their transition functions are the same.
- Update transition functions to take account of merged states. Repeat.

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.

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.

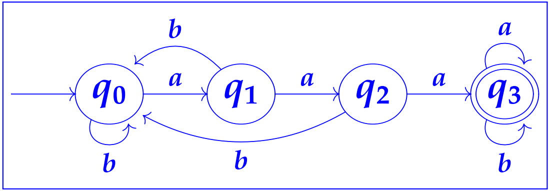
(b) Conversely, every regular language is the form $L(r)$ for some regular expression r.

The not so fun side of Kleene's Theorem

Example of a regular language

Recall the example DFA we used earlier:

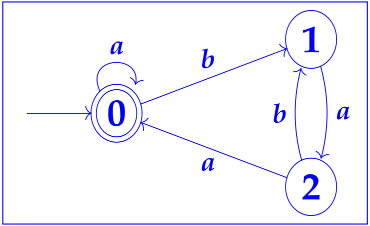
$M \triangleq$



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

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 $\emptyset$).

Prove this Lemma By induction on # of elements in S
Also take care to examine case where $q = q'$!

Base case $S = \emptyset$

Given states $q, q' \in M$, if

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

holds for just $a = a_1, a_2, \ldots, a_k$ then can define

$$r^{\emptyset}_{q,q'} \triangleq \begin{cases} a = a_1|a_2|\ldots|a_k & \text{if } q \neq q' \\ a = a_1|a_2|\ldots|a_k|\epsilon & \text{if } q = q' \end{cases}$$

Induction Step:

- S has $n + 1$ elements.

Induction Step:

- S has $n+1$ elements.
- pick some $q_0 \in S$

Induction Step:

- S has $n+1$ elements.
- pick some $q_0 \in S$
- consider $S^- = S \setminus \{q_0\}$ (S without the state q_0)

Induction Step:

- S has $n+1$ elements.
- pick some $q_0 \in S$
- consider $S^- = S \setminus \{q_0\}$ (S without the state q_0)
- can apply induction hypoth to S^- since S^- has n elements

Induction Step:

- S has $n+1$ elements.
- pick some $q_0 \in S$
- consider $S^- = S \setminus \{q_0\}$ (S without the state q_0)
- can apply induction hypoth to S^- since S^- has n elements

Can we express $r^S_{q,q'}$ in terms of things only depending on S^-?

What's in $r^S_{q,q'}$?

- we might be able to get from q to q' through S avoiding q_0, and

What's in $r^S_{q,q'}$?

- we might be able to get from q to q' through S avoiding q_0, and
- we might be able to get from q to q_0, then from q_0 back to itself an arbitrary number of times, then to q'

What's in $r^S_{q,q'}$?

- we might be able to get from q to q' through S avoiding q_0, and
- we might be able to get from q to q_0, then from q_0 back to itself an arbitrary number of times, then to q'

For the first of these we have $r^{S^-}_{q,q'}$ by hypothesis. (If there is no path, this will be $\emptyset$)

What's in $r^S_{q,q'}$?

- we might be able to get from q to q' through S avoiding q_0, and
- we might be able to get from q to q_0, then from q_0 back to itself an arbitrary number of times, then to q'

For the first of these we have $r^{S^-}_{q,q'}$ by hypothesis. (If there is no path, this will be $\emptyset$)

For the second we have $r^{S^-}_{q,q_0} \; [r^{S^-}_{q_0,q_0}]^* \; r^{S^-}_{q_0,q'}$

$$r^{S}_{q,q'} = r^{S^-}_{q,q'} | (r^{S^-}_{q,q_0} [r^{S^-}_{q_0,q_0}]^* r^{S^-}_{q_0,q'})$$