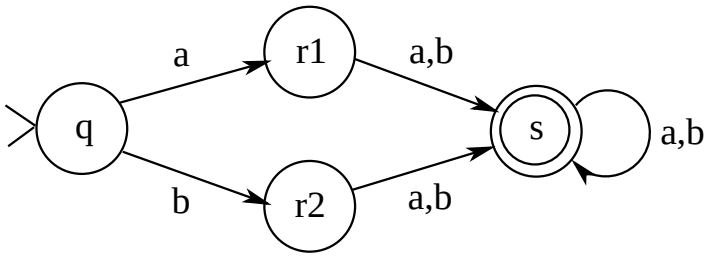
1 Minimizing Finite Automata

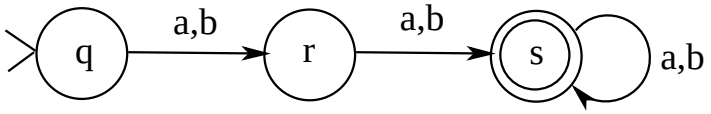
Outline of this section:

1. Define strings equivalent with respect to a language L. This is notated as $x \approx_L y$. This is defined by $x \approx_L y$ iff $\{z : xz \in L\} = \{z : yz \in L\}$. If L is regular then $\approx_L$ has finitely many equivalence classes, and vice versa.
2. Define strings equivalent with respect to a deterministic finite automaton M. This is written $x \sim_M y$. Two strings are equivalent with respect to M if they cause M to end up in the same state.
3. Use $\approx_L$ to
   (a) Characterize regular languages. (L is regular iff $\approx_L$ has finitely many equivalence classes.)
   (b) Compute the smallest number of states in any deterministic finite automaton recognizing L. (It is equal to the number of equivalence classes of $\approx_L$.)
4. It is possible to show a language L non-regular by showing that $\approx_L$ has infinitely many equivalence classes.
5. Define the equivalence of two states in a deterministic finite automaton. $p \equiv q$ in M if $L(M_p) = L(M_q)$ where M_x is M with x as the start state.
6. Compute which states of an automaton M are equivalent; then collapse equivalent states and delete unreachable states to minimize M.

Here is an example of a non-minimal finite automaton:



Here is an equivalent minimal automaton:



1.1 Equivalence with Respect to L

$$x \approx_L y \text{ iff } \{z : xz \in L\} = \{z : yz \in L\}$$

Note that L need not be regular for this definition. The equivalence relation $\approx_L$ can also be defined this way:

> If there is a z such that $xz \in L$ and $yz \notin L$ then $x \not\approx_L y$
> If there is a z such that $xz \notin L$ and $yz \in L$ then $x \not\approx_L y$
> Otherwise $x \approx_L y$.

1.1.1 Example

Let L be $\{ab, ac, bb, bc\}$.

x	$\{z : xz \in L\}$
a	$\{b, c\}$
b	$\{b, c\}$
c	ϕ
ab	$\{\epsilon\}$
ac	$\{\epsilon\}$
ϵ	L
abc	ϕ

Thus $a \approx_L b$ and $ab \approx_L ac$ but $a \not\approx_L c$ and $b \not\approx_L c$, for example. There are four equivalence classes, corresponding to the four values of $\{z : xz \in L\}$ as x varies.

Let L be $\{w \in \{0,1\}^* : |w| \text{ is even}\}$.

x	$\{z : xz \in L\}$
0	odd length strings
1	odd length strings
00	even length strings
01	even length strings
101	odd length strings

Thus $0 \approx_L 1$, $00 \approx_L 01$, $0 \approx_L 101$, $0 \not\approx_L 00$, and $1 \not\approx_L 01$, for example. There are two equivalence classes.

What is an equivalence class?

Definition 1.1 (Equivalence Class) *Given a set S and an equivalence relation R, the equivalence classes of S are*

- *disjoint subsets $S_1, S_2, \ldots$ of S*
- *whose union is S*
- *and such that if $x, y \in S_i$ then xRy*
- *but if x and y are in different subsets S_i and S_j for $i \neq j$ then it is not true that xRy.*

For the relation $\approx_L$, there is one equivalence class for each value of $\{z : xz \in L\}$.

Here is a way to think of equivalence classes:

List all the elements of the set S, say, $x_1, x_2, x_3, \ldots$.

- If some x_i is not equivalent to any element seen so far, then x_i starts a new equivalence class.
- If some x_j is equivalent to some x_i seen earlier, then x_j is in the same equivalence class as x_i.
- The number of equivalence classes is just the number of sets found this way in the limit.

If L is regular, then one can test if $x \approx_L y$ this way:

- Let M be a minimal deterministic finite automaton recognizing L.
- Then $x \approx_L y$ iff x and y both drive M from the start state to the same state of M.

This can be an easy way to test if $x \approx_L y$ if you can guess M.

1.2 Problems

Do these in class.

What are the equivalence classes for a^*b^*?

- Find them using the listing idea given above.
- How many equivalence classes are there?
- Make each one into a state and show how one can construct a minimal deterministic finite automaton from them.
- Explain how to choose the start state and accepting states and how to draw the arrows.
- The resulting automaton is minimal for this language.

How about for $\{a^n b^n : n \geq 0\}$? What are the equivalence classes?

1.3 A convenient definition

If M is a deterministic or nondeterministic finite state automaton, write

$$s \xrightarrow{a}_M t$$

if M, in state s, reading a symbol $a \in \Sigma$, can end up in state t.

- That is, $\delta(s, a) = t$ if M is deterministic, and $(s, a, t) \in \Delta$ if M is nondeterministic.

Also, write

$$s \xrightarrow{w}{}^*_M t$$

if M, in state s, reading a string $w \in \Sigma^*$, can end up in state t.

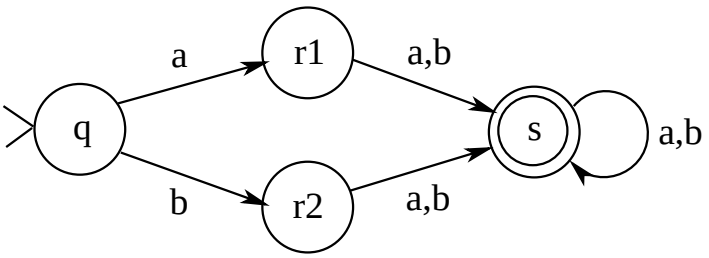
- If M is deterministic, then t is determined by s and w.
- If M is nondeterministic, then there could be more than one such t, one, or none, for a given s and w.

1.4 Equivalence with respect to M

Definition 1.2 (2.5.2)

- *Two strings $x, y \in \Sigma^*$ are* equivalent with respect to M, *written $x \sim_M y$, if $s \xrightarrow{x}{}^*_M t$ and $s \xrightarrow{y}{}^*_M u$ implies $t = u$, where s is the start state of M.*
- *That is, when M reads the string x starting in the start state, it ends up in the same state as when it reads the string y starting in the start state.*

Consider again this automaton:



- The strings a and b are not equivalent with respect to M, because they cause M to end up in states r_1 and r_2, respectively, starting at the start state q.
- However, the strings aa and ba are equivalent with respect to M, because both strings cause M to end up in state s.
- Also, aa and aaa are equivalent with respect to M.

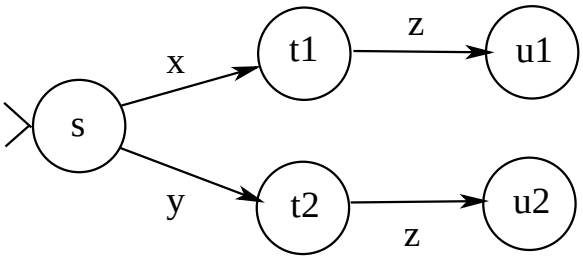
Theorem 1.1 (2.5.1) *For any deterministic finite automaton $M = (K, \Sigma, \delta, s, F)$, and any strings $x, y \in \Sigma^*$, if $x \sim_M y$ then $x \approx_{L(M)} y$.*

Proof: Suppose $x \sim_M y$. We want to show that $x \approx_{L(M)} y$, that is, for all z, $xz \in L(M)$ iff $yz \in L(M)$.

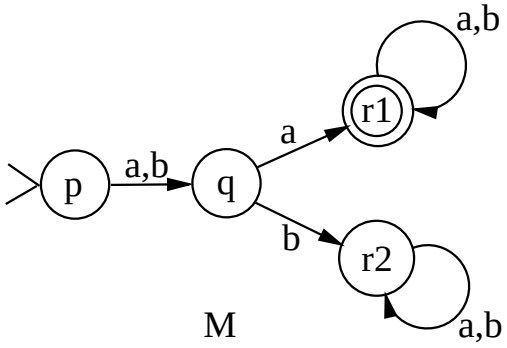
- Suppose $s \xrightarrow{x}{}^*_M t_1$ and $s \xrightarrow{y}{}^*_M t_2$, where s is the start state of M.
- Because $x \sim_M y$, $t_1 = t_2$.

- Now, for some states u_1 and u_2, $s \stackrel{xz}{\rightarrow}{}^*_M u_1$ and $s \stackrel{yz}{\rightarrow}{}^*_M u_2$.
- However, in reading xz, M will first read x and go to t_1.
- Then, starting in state t_1, M will read z and go to u_1.
- In reading yz, M will first read y and go to t_2, which equals t_1.
- Then, starting in state t_2, M will read z and go to state u_2.
- Because $t_1 = t_2$ and M is deterministic, $u_1 = u_2$.
- Thus xz is accepted iff u_1 is an accepting state, iff u_2 is an accepting state, iff yz is accepted.
- So $xz \in L(M)$ iff $yz \in L(M)$.
- Therefore $x \approx_{L(M)} y$.

Diagram:



Example:



- In this example, $a \sim_M b$ because both a and b lead from the start state to state q.
- Now, we claim that also $a \approx_{L(M)} b$.
- This means that if $az \in L(M)$ then $bz \in L(M)$ and vice versa.
- Let's look at some z to see why this is true.
- Consider $z = a$.
- Then $aa \in L(M)$ because a leads from the start state to state q and then a leads from state q to state r_1, so aa leads to the state r_1 which is an accepting state of M.
- Is $bz \in L(M)$ also?
- Yes, because b leads from the start state to state q and then a leads from state q to state r_1.
- Consider $z = b$.
- Then $ab \notin L(M)$ because a leads to state q and then b leads to state r_2 which is not an accepting state.
- Is $bz \in L(M)$?
- No because b leads to state q and then b leads to state r_2 which is not an accepting state.
- The same holds for any z so $az \in L(M)$ iff $bz \in L(M)$, so $a \approx_{L(M)} b$.

In general,

- if z leads from state q to an accepting state of M, then $az \in L(M)$ and $bz \in L(M)$.
- If z leads from state q to a non-accepting state of M, then $az \notin L(M)$ and $bz \notin L(M)$.
- So $az \in L(M)$ iff $bz \in L(M)$, so $a \approx_{L(M)} b$.

This theorem implies that:

> any deterministic finite automaton M recognizing L has to have at least as many states as the number of equivalence classes of the relation $\approx_L$.

Why? If there were more equivalence classes than states, then there must be some state q of M and two strings x, y in Σ^* such that $x \not\approx_L y$ but x and y both end up in state q, so that $x \sim_M y$, This is impossible by the theorem.

Also,

> If L is regular then $\approx_L$ has finitely many equivalence classes.

Why?

- If L is regular then there is a finite automaton M recognizing L.
- M has finitely many states.
- But the number of states of M is the same as the number of equivalence classes of $\sim_M$ (or larger if some states are unreachable), so the number of equivalence classes of $\sim_M$ is finite.
- This is at least as large or larger than the number of equivalence classes of $\approx_L$, which therefore must be finite.

If x is an element of Σ^* then let $[x]$ be the *equivalence class of* x, that is, the set of y such that $x \approx_L y$.

Theorem 1.2 (2.5.2, Myhill-Nerode Theorem) *Let $L \subseteq \Sigma^*$ be a regular language. Then there is a minimal deterministic finite automaton M which has a number of states equal to the number of equivalence classes of the relation $\approx_L$.*

Proof: The states of M are the equivalence classes of $\approx_L$. The start state of M is $[e]$. The accepting states of M are the set of equivalence classes that are subsets of L. Define $\delta([x], a)$ to be $[xa]$ for $x \in \Sigma^*$ and $a \in \Sigma$.

The detailed proof is in the text.

This quote is from CACM May 2020 vol. 63 No. 5 p. 82:

> The Myhill-Nerode theorem, "one of the conceptual gems of theoretical computer science" according to Rosenberg, offers a complete characterization of the notion of state, via basic algebraic properties defined only on input/output behvior.

Construct this automaton for $L = \mathcal{L}(a^*b^*)$.

So just from L itself, we can tell how many states there must be in a minimal deterministic finite automaton for L.

Thus:

> If L is regular, then the minimal deterministic finite state automaton recognizing L has a number of states equal to the number of equivalence classes of $\approx_L$.

Also, this theorem gives a systematic way to construct a finite automaton M recognizing L, given L.

- Each equivalence class of $\approx_L$ is a state of M.
- That is, each possible value for $\{z : xz \in L\}$ is a state of M.
- The start state is the set of all strings x such that $\{z : xz \in L\} = L$, that is, it is the equivalence class of the empty string..
- The accepting states are the equivalence classes that are subsets of L.

This approach can be used, for example, to construct a finite automaton recognizing the set of w such that w has at least 2 a's and an odd number of b's.

Corollary 1.1 *A language is regular iff $\approx_L$ has finitely many equivalence classes.*

Proof: If L is regular then L is recognized by some deterministic automaton M, so $\approx_L$ has finitely many equivalence classes. If $\approx_L$ has finitely many euqivalence classes, then by the Myhill-Nerode theorem, L is regular.

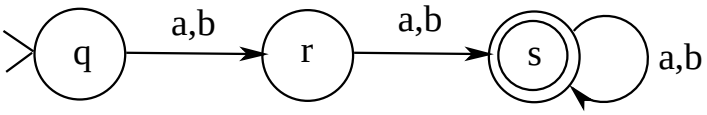
This gives another method to show a language L is not regular:

> A language L is not regular if $\approx_L$ has infinitely many equivalence classes.

We want to minimize finite automata. However, it is possible to verify that a deterministic finite automaton is minimal, as follows:

> A deterministic finite automaton M is minimal if
>
> - for each pair p, q of states of M,
> - there is a string $w \in \Sigma^*$ such that
> - M accepts w starting from p,
> - but M does not accept w starting from q,
> - or vice versa.
> - Also, all states must be reachable from the start state.

Thus we may be able to guess an automaton M and verify that it is minimal without going through the whole minimization algorithm. Do this on the following automaton:



1.5 Equivalent states of a finite automaton

Definition 1.3 *If $M = (K, \Sigma, \delta, s, F)$ is a deterministic finite automaton, then for two states p, q of M, $p \equiv q$ if the following is true:*

For all $x \in \Sigma^*$,

- *when M starts in state p and reads x, it ends in an accepting state,*

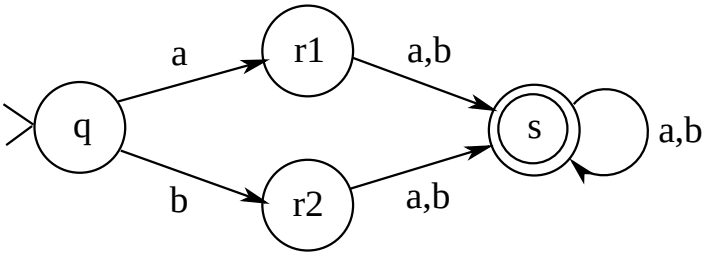
- *if and only if,*
- *when M starts in state q and reads x, it ends in an accepting state.*

Here is another definition of the same concept:

Definition 1.4

- *Suppose $M = (K, \Sigma, \delta, s, F)$ is a deterministic finite automaton.*
- *Then for a state $q \in K$, M_q is the automaton $(K, \Sigma, \delta, q, F)$.*
- *Thus q has been made into the start state.*
- *Then two states p and q of M are equivalent if the automata M_p and M_q are equivalent, that is, $L(M_p) = L(M_q)$*

Consider again this automaton:



The states r_1 and r_2 of this automaton are equivalent. No other pair of states of this automaton is equivalent.

To minimize a finite automaton M, we do the following:

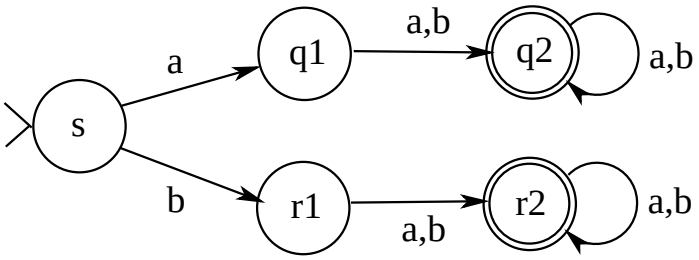
1. Compute the equivalence relation $\equiv$ on states of M.
2. Collapse all pairs p, q of states of M such that $p \equiv q$.
3. Delete all states of M that are not reachable from the start state.

The text gives an iterative algorithm for computing the equivalence relation M. I give a different iterative algorithm based on *colorings* of the states of M.

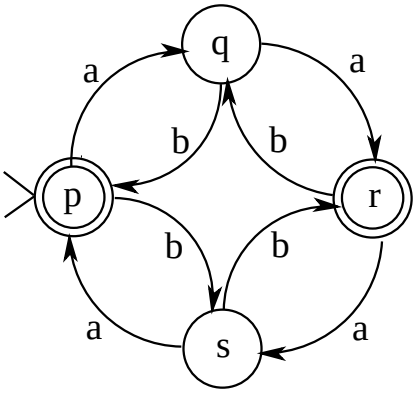
Suppose $M = (K, \Sigma, \delta, s, F)$ and $\Sigma = \{a_1, \ldots, a_n\}$.

1. The 0-coloring of M colors all states in F one color and all states in $K - F$ another color.
2. Suppose the i-coloring of M is defined. The *i-color co-ordinates* of state q are $(c_q, c_1, c_2, \ldots, c_n)$ where c_q is the i-color of q and c_k is the i-coloring of $\delta(q, a_k)$.
3. The $i + 1$ coloring of M is defined so that two states q, r have the same i+1 color if and only if they have the same i-color co-ordinates.
4. The coloring terminates when there are the same number of i colors as $i + 1$ colors, for some i.
5. When the coloring terminates at coloring i, then two states p and q satisfy $p \equiv q$ if and only if p and q have the same i coloring.

For an example, see Handout 3 on the course web page, which minimizes this automaton:



Also, do it on the following automaton:



2 Algorithms for Finite Automata

- Convert a nondeterministic automaton to a deterministic automaton
- Generate a regular expression from an automaton
- Minimize a finite automaton
- Test equivalence of two deterministic finite automata
- Test equivalence of two nondeterministic finite automata
- For a regular language L and a string x, test if $x \in L$
- For a regular expression E and a string x, test if $x \in \mathcal{L}(E)$
- For a nondeterministic automaton M and a string x, test if $x \in L(M)$
- Given strings w and x, to test if x is a substring of w
- Test if two regular expressions are equivalent

Which of these are polynomial and which are exponential? What are the time bounds? It also depends on how regular expressions are represented, whether linearly or as directed acyclic graphs with pointers to common subexpressions.