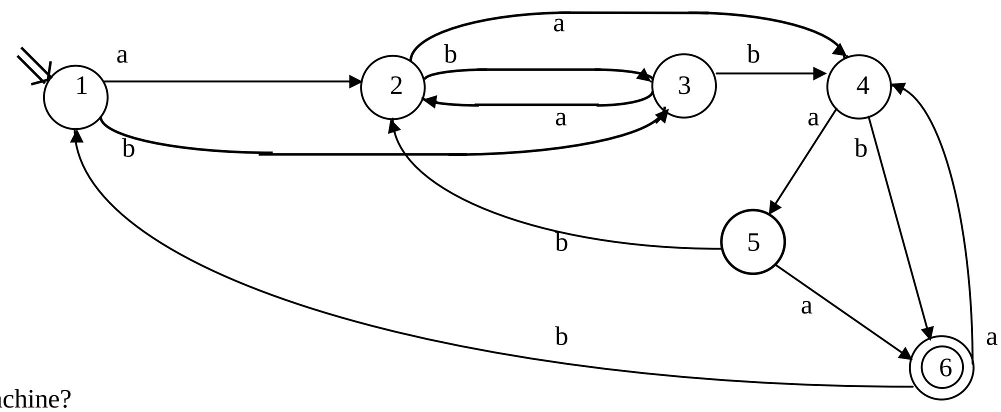
State Minimization for DFAs

Read K & S 2.7
Do Homework 10.

State Minimization

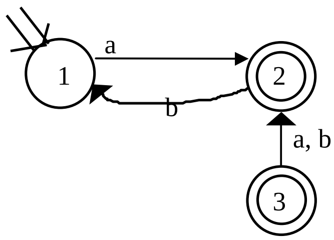
Consider:



Is this a minimal machine?

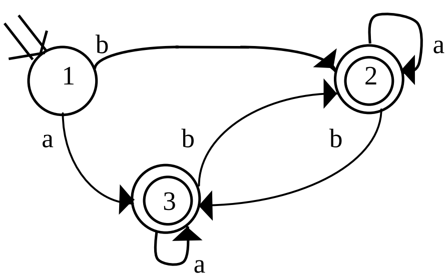
State Minimization

Step (1): Get rid of unreachable states.



State 3 is unreachable.

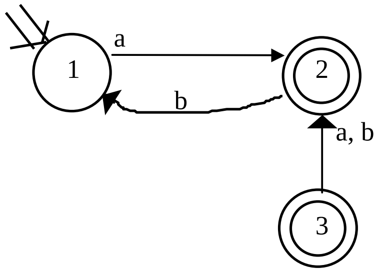
Step (2): Get rid of redundant states.



States 2 and 3 are redundant.

Getting Rid of Unreachable States

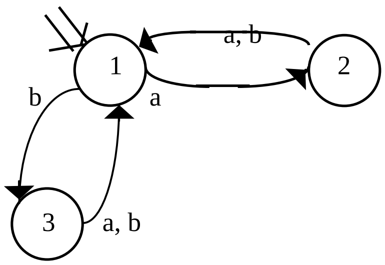
We can't easily find the unreachable states directly. But we can find the reachable ones and determine the unreachable ones from there. An algorithm for finding the reachable states:



Getting Rid of Redundant States

Intuitively, two states are equivalent to each other (and thus one is redundant) if all strings in Σ* have the same fate, regardless of which of the two states the machine is in. But how can we tell this?

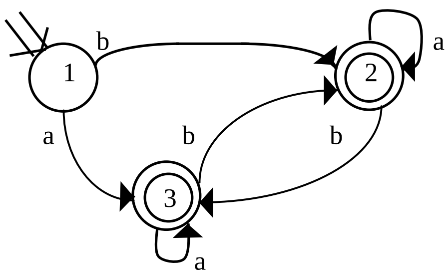
The simple case:



Two states have identical sets of transitions out.

Getting Rid of Redundant States

The harder case:



The outcomes are the same, even though the states aren't.

Finding an Algorithm for Minimization

Capture the notion of equivalence classes of strings with respect to a language.

Capture the (weaker) notion of equivalence classes of strings with respect to a language and a particular FSA.

Prove that we can always find a deterministic FSA with a number of states equal to the number of equivalence classes of strings.

Describe an algorithm for finding that deterministic FSA.

Defining Equivalence for Strings

We want to capture the notion that two strings are equivalent with respect to a language L if, no matter what is tacked on to them on the right, either they will both be in L or neither will. Why is this the right notion? Because it corresponds naturally to what the states of a recognizing FSM have to remember.

Example:

(1)	a	b	|	b	a	b
(2)	b	a	|	b	a	b

Suppose L = {w ∈ {a,b}* : |w| is even}. Are (1) and (2) equivalent?

Suppose L = {w ∈ {a,b}* : every a is immediately followed by b}. Are (1) and (2) equivalent?

Defining Equivalence for Strings

If two strings are equivalent with respect to L, we write $x \approx_L y$. Formally, $x \approx_L y$ if, $\forall z \in \Sigma^*$,

$xz \in L$ iff $yz \in L$.

Notice that $\approx_L$ is an equivalence relation.

Example:

$\Sigma = \{a, b\}$

$L = \{w \in \Sigma^* :$ every a is immediately followed by b $\}$

ε	aa	bbb
a	bb	baa
b	aba	
	aab	

The equivalence classes of $\approx_L$:

$|\approx_L|$ is the number of equivalence classes of $\approx_L$.

Another Example of $\approx_L$

$\Sigma = \{a, b\}$

$L = \{w \in \Sigma^* : |w|$ is even$\}$

ε	bb	aabb
a	aba	bbaa
b	aab	aabaa
aa	bbb	
	baa	

The equivalence classes of $\approx_L$:

Yet Another Example of $\approx_L$

$\Sigma = \{a, b\}$

$L = aab^*a$

ε	ba	aabb
a	bb	aabaa
b	aaa	aabbba
aa	aba	aabbaa
ab	aab	
	bab	

The equivalence classes of $\approx_L$:

An Example of $\approx_L$ Where All Elements of L Are Not in the Same Equivalence Class

$\Sigma = \{a, b\}$

$L = \{w \in \{a, b\}^* :$ no two adjacent characters are the same$\}$

ε	bb	aabaa
a	aba	aabbba
b	aab	aabbaa
aa	baa	
	aabb	

The equivalence classes of $\approx_L$:

Is $|\approx_L|$ Always Finite?

$\Sigma = \{a, b\}$
$L = a^n b^n$

ε	aa	aaaa
a	aba	aaaaa
b	aaa	

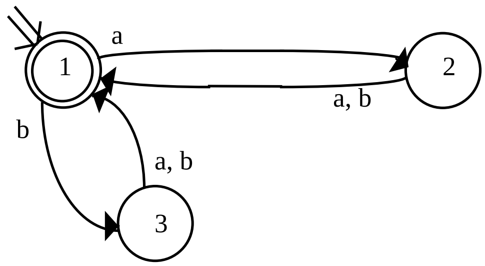
The equivalence classes of $\approx_L$:

Bringing FSMs into the Picture

$\approx_L$ is an ideal relation.

What if we now consider what happens to strings when they are being processed by a real FSM?

$\Sigma = \{a, b\}$ $L = \{w \in \Sigma^* : |w| \text{ is even}\}$



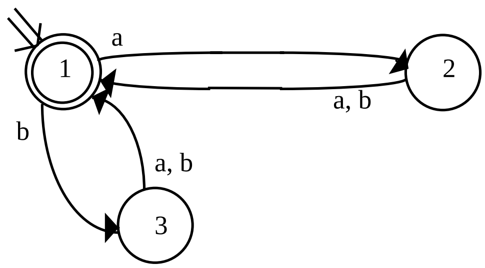
Define $\sim_M$ to relate pairs of strings that drive M from s to the same state.

Formally, if M is a deterministic FSM, then $x \sim_M y$ if there is some state q in M such that $(s, x) \vdash^*_M (q, \varepsilon)$ and $(s, y) \vdash^*_M (q, \varepsilon)$.

Notice that M is an equivalence relation.

An Example of ~M

$\Sigma = \{a, b\}$ $L = \{w \in \Sigma^* : |w| \text{ is even}\}$

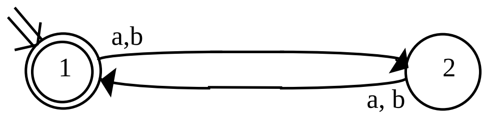


ε	bb	aabb
a	aba	bbaa
b	aab	aabaa
aa	bbb	
	baa	

The equivalence classes of $\sim_M$: $|\sim_M| =$

Another Example of ~M

$\Sigma = \{a, b\}$ $\qquad$ $L = \{w \in \Sigma^* : |w| \text{ is even}\}$



ε	bb	aabb
a	aba	bbaa
b	aab	aabaa
aa	bbb	
	baa	

The equivalence classes of $\sim_M$: $\qquad$ $|\sim_M| =$

The Relationship Between $\approx_L$ and $\sim_M$

$\approx_{L:}$ [ε, aa, bb, aabb, bbaa] $\quad$ |w| is even
[a, b, aba, aab, bbb, baa, aabaa] |w| is odd

$\sim_M$, 3 state machine:
- q1: [ε, aa, bb, aabb, bbaa] $\quad$ |w| is even
- q2: [a, aba, baa, aabaa] $\quad$ $(ab \cup ba \cup aa \cup bb)^*a$
- q3: [b, aab, bbb] $\quad$ $(ab \cup ba \cup aa \cup bb)^*b$

$\sim_M$, 2 state machine:
- q1: [ε, aa, bb, aabb, bbaa] $\quad$ |w| is even
- q2: [a, b, aba, aab, bbb, baa, aabaa] |w| is odd

$\sim_M$ is a refinement of $\approx_L$.

The Refinement

$\approx_L$	[even length]	[odd length]		(S)
$\sim_M$ (3 state)	[even length]	[odd ending in a]	[odd ending in b]	(R)

An equivalence relation R is a refinement of another one S iff

$$xRy \rightarrow xSy$$

In other words, R makes all the same distinctions S does, plus possibly more.

$|R| \geq |S|$

$\sim_M$ is a Refinement of $\approx_L$.

Theorem: For any deterministic finite automaton M and any strings x, y $\in \Sigma^*$, if x $\sim_M$ y, then x $\approx_L$ y.

Proof: If x $\sim_M$ y, then x and y drive m to the same state q. From q, any continuation string w will drive M to some state r. Thus xw and yw both drive M to r. Either r is a final state, in which case they both accept, or it is not, in which case they both reject. But this is exactly the definition of $\approx_L$.

Corollary: $|\sim_M| \geq |\approx_L|$.

Going the Other Way

When is this true?

If x $\approx_{L(M)}$ y then x $\sim_M$ y.

Finding the Minimal FSM for L

What's the smallest number of states we can get away with in a machine to accept L?

Example: $L = \{w \in \Sigma^* : |w| \text{ is even}\}$

The equivalence classes of $\approx_L$:

Minimal number of states for M(L) =

This follows directly from the theorem that says that, for any machine M that accepts L, $|\sim_M|$ must be at least as large as $|\approx_L|$.

Can we always find a machine with this minimal number of states?

The Myhill-Nerode Theorem

Theorem: Let L be a regular language. Then there is a deterministic FSA that accepts L and that has precisely $|\approx_L|$ states.
Proof: (by construction)
M = K states, corresponding to the equivalence classes of $\approx_L$.
s = [ε], the equivalence class of ε under $\approx_L$.
F = {[x] : x $\in$ L}
δ([x], a) = [xa]

For this construction to prove the theorem, we must show:

1. K is finite.
2. δ is well defined, i.e., δ([x], a) = [xa] is independent of x.
3. L = L(M)

The Proof

(1) K is finite.
Since L is regular, there must exist a machine M, with $|\sim_M|$ finite. We know that

$$|\sim_M| \geq |\approx_L|$$

Thus $|\approx_L|$ is finite.
(2) δ is well defined.
This is assured by the definition of $\approx_L$, which groups together precisely those strings that have the same fate with respect to L.

The Proof, Continued

(3) L = L(M)
Suppose we knew that $([x], y) \mid\!-_M^* ([xy], \varepsilon)$.
Now let [x] be [ε] and let s be a string in Σ*.
Then

$$([\varepsilon], s) \mid\!-_M^* ([s], \varepsilon)$$

M will accept s if $[s] \in F$.
By the definition of F, $[s] \in F$ iff all strings in [s] are in L.
So M accepts precisely the strings in L.

The Proof, Continued

Lemma: $([x], y) \mid\!-_M^* ([xy], \varepsilon)$
By induction on |y|:
Trivial if |y| = 0.
Suppose true for |y| = n.
Show true for |y| = n+1

Let y = y'a, for some character a. Then,

|y'| = n

$([x], y'a) \mid\!-_M^* ([xy'], a)$	(induction hypothesis)
$([xy',] a) \mid\!-_M^* ([xy'a], \varepsilon)$	(definition of δ)
$([x], y'a) \mid\!-_M^* ([xy'a], \varepsilon)$	(trans. of $\mid\!-_M^*$)
$([x], y) \mid\!-_M^* ([xy], \varepsilon)$	(definition of y)

Another Version of the Myhill-Nerode Theorem

Theorem: A language is regular iff $|\approx_L|$ is finite.
Example:

Consider: $L = a^nb^n$
a, aa, aaa, aaaa, aaaaa …
Equivalence classes:

Proof:
Regular → $|\approx_L|$ *is finite*: If L is regular, then there exists an accepting machine M with a finite number of states N. We know that $N \geq |\approx_L|$. Thus $|\approx_L|$ is finite.

$|\approx_L|$ *is finite* → *regular*: If $|\approx_L|$ is finite, then the standard DFSA M_L accepts L. Since L is accepted by a FSA, it is regular.

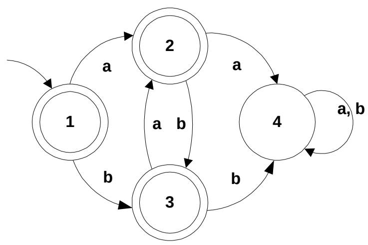
Constructing the Minimal DFA from $\approx_L$

$\Sigma = \{a, b\}$
L = {w ∈ {a, b}* : no two adjacent characters are the same}

The equivalence classes of $\approx_L$:

1: [ε]	ε
2: [a, ba, aba, baba, ababa, ...]	(b∪ε)(ab)*a
3: [b, ab, bab, abab, ...]	(a∪ε)(ba)*b
4: [bb, aa, bba, bbb, ...]	the rest

- Equivalence classes become states
- Start state is [ε]
- Final states are all equivalence classes in L
- δ([x], a) = [xa]



Using Myhill-Nerode to Prove that L is not Regular

L = {a^n : n is prime}

Consider: ε
a
aa
aaa
aaaa

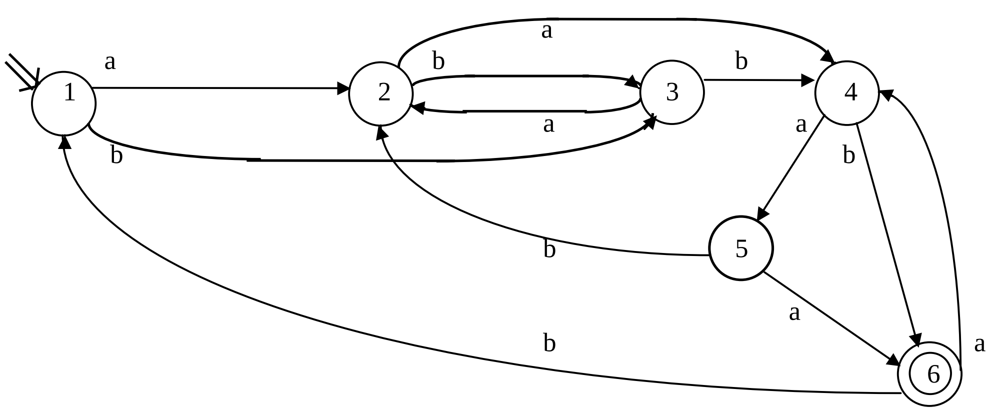
Equivalence classes:

So Where Do We Stand?

1. We know that for any regular language L there exists a minimal accepting machine M_L.
2. We know that |K| of M_L equals $|\approx_L|$.
3. We know how to construct M_L from $\approx_L$.

But is this good enough?

Consider:



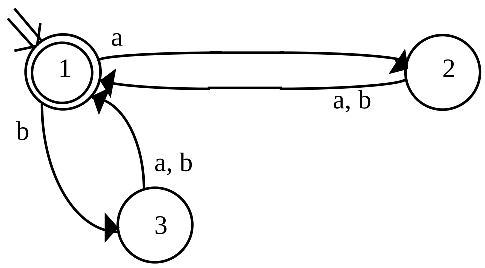
Constructing a Minimal FSA Without Knowing $\approx_L$

We want to take as input any DFSA M' that accepts L, and output a minimal, equivalent DFSA M.

What we need is a definition for "equivalent", i.e., mergeable states.

Define $q \equiv p$ iff for all strings $w \in \Sigma^*$, either w drives M to an accepting state from both q and p or it drives M to a rejecting state from both q and p.

Example:
$\Sigma = \{a, b\}$ L = $\{w \in \Sigma^* : |w| \text{ is even}\}$



Constructing $\equiv$ as the Limit of a Sequence of Approximating Equivalence Relations $\equiv_n$

(Where n is the length of the input strings that have been considered so far)

We'll consider input strings, starting with ε, and increasing in length by 1 at each iteration. We'll start by way overgrouping states. Then we'll split them apart as it becomes apparent (with longer and longer strings) that their behavior is not identical.

Initially, $\equiv_0$ has only two equivalence classes: [F] and [K - F], since on input ε, there are only two possible outcomes, accept or reject.

Next consider strings of length 1, i.e., each element of Σ. Split any equivalence classes of $\equiv_0$ that don't behave identically on all inputs. Note that in all cases, $\equiv_n$ is a refinement of $\equiv_{n-1}$.

Continue, until no splitting occurs, computing $\equiv_n$ from $\equiv_{n-1}$.

Constructing $\equiv$, Continued

More precisely, for any two states p and $q \in K$ and any $n \geq 1$, $q \equiv_n p$ iff:

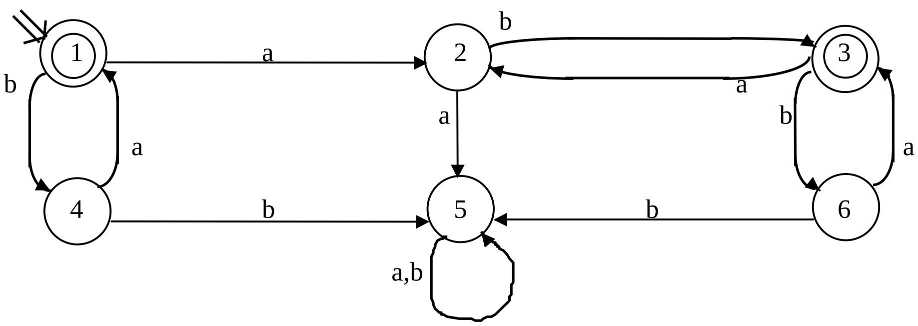
1. $q \equiv_{n-1} p$, AND
2. for all $a \in \Sigma$, $\delta(p, a) \equiv_{n-1} \delta(q, a)$

The Construction Algorithm

The equivalence classes of $\equiv_0$ are F and K-F.
Repeat for n = 1, 2, 3 …
  For each equivalence class C of $\equiv_{n-1}$ do
    For each pair of elements p and q in C do
      For each a in Σ do
        See if $\delta(p, a) \equiv_{n-1} \delta(q, a)$
      If there are any differences in the behavior of p and q, then split them and create a new equivalence class.
Until $\equiv_n = \equiv_{n-1}$. ≡ is this answer. Then use these equivalence classes to coalesce states.

An Example

Σ = {a, b}



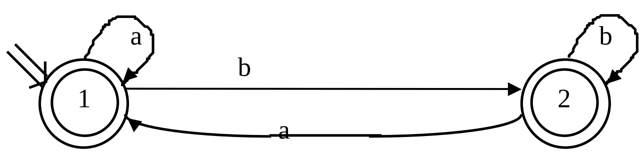
$\equiv_0$ =

$\equiv_1$ =

$\equiv_2$ =

Another Example

(a*b*)*



$\equiv_0$ =

$\equiv_1$ =

Minimal machine:

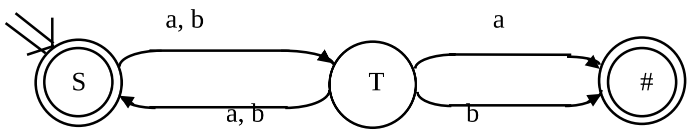
Another Example

Example: L={w ∈ {a, b}* : |w| is even}

$((aa) \cup (ab) \cup (ba) \cup (bb))^*$

$S \rightarrow \varepsilon$
$S \rightarrow aT$
$S \rightarrow bT$

$T \rightarrow a$
$T \rightarrow b$
$T \rightarrow aS$
$T \rightarrow bS$



Convert to deterministic:
S = {s}
δ =

Another Example, Continued

Minimize:

$\equiv_0$ =

$\equiv_1$ =

Minimal machine: