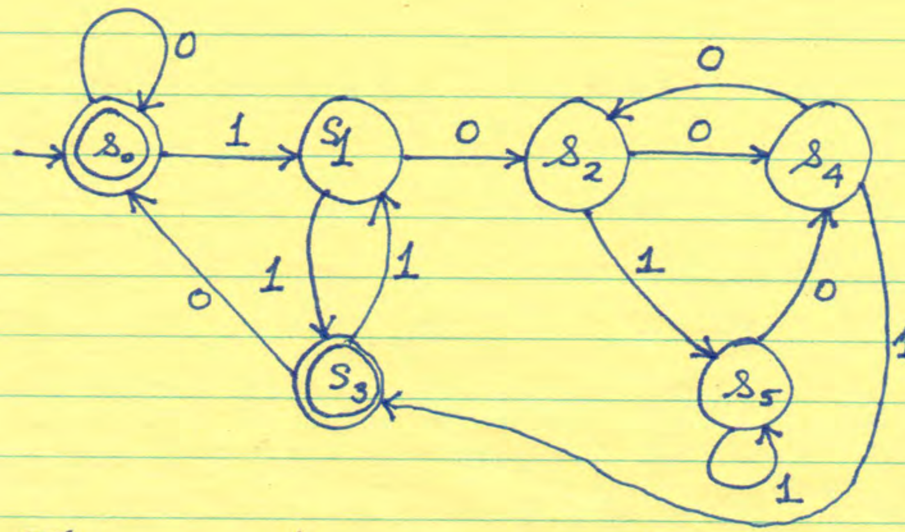
Minimization of DFAs:



This machine accepts binary strings that represent numbers which are divisible by 3. But it keeps track of remainders mod 6. Thus it has twice as many states as it needs.

If we start with the above machine, can we "shrink" it down to the 3-state machine we had earlier? Well, do we need both s_0 and s_3? Once the DFA is in state s_3 any transition takes it to the same state as the same transition from s_0:

$$\delta(s_0, 0) = s_0 = \delta(s_3, 0)$$
$$\delta(s_0, 1) = s_1 = \delta(s_3, 1).$$

So nothing in the subsequent behaviour of the machine can tell the difference between s_0 & s_3. Furthermore s_0, s_3 are both accept states. Thus as far as language recognition is concerned they are equivalent. We should try to define an equivalence relation based on this idea. The key insight: we can build a smaller machine by using equivalence classes of states.

<u>Def</u> Given a DFA $M = (S, s_0, \delta, F)$ over alphabet Σ we say $p, q \in S$ are <u>equivalent</u>, and write $p \approx q$, if

$$\forall x \in \Sigma^* \quad \delta^*(p,x) \in F \Leftrightarrow \delta^*(q,x) \in F.$$

<u>Intuition</u> : If we made p the start state but otherwise kept the same we would recognize (accept) the same language as if we made q the start state.

<u>Remark</u> When are p, q not equivalent?
$p \not\approx q$ means $\exists x \in \Sigma^*$ s.t. $(\delta^*(p,x) \in F$ & $\delta^*(q,x) \notin F)$ OR $(\delta^*(p,x) \notin F$ & $\delta^*(q,x) \in F)$.

<u>Observation</u> $\approx$ is an equivalence relation (Check it!).
We write $[p]$ for the equivalence class of p.

<u>Lemma A</u> $p \approx q \Rightarrow \forall a \in \Sigma \ \delta(p,a) \approx \delta(q,a)$

<u>Proof</u> Suppose $\delta^*(\delta(p,a),x) \in F$
then $\delta^*(p, ax) \in F$.

By assumption $p \approx q$ we know $\delta^*(q, ax) \in F$
or $\delta^*(\delta(q,a), x) \in F$.
Similarly for the case $\delta^*(\delta(p,a), x) \notin F$.
Since nothing was assumed about x, this holds for all x. Thus $\delta(p,a) \approx \delta(q,a)$. ∎

<u>REMARK</u> : $p \approx q$ can be written $[p] = [q]$. So the lemma says $[p] = [q] \Rightarrow [\delta(p,a)] = [\delta(q,a)]$.

We define a new machine $M' = (S', s_0', \delta', F')$

S' = equivalence classes of S $\quad (S/\approx)$

$s_0' = [s_0]$

$\delta'([p], a) = [\delta(p,a)]$ [Well defined]

$F' = \{[s] \mid s \in F\}$

<u>Lemma B</u> $p \in F$ & $p \approx q \Rightarrow q \in F$ [Do it yourself].

<u>Lemma C</u> $\forall \omega \in \Sigma^* \quad \delta'^*([p], \omega) = [\delta^*(p, \omega)]$.

<u>Proof</u> Induction on ω

<u>BASE</u> $\omega = \varepsilon \quad \delta'^*([p], \varepsilon) = [p] = [\delta^*(p, \varepsilon)]$

<u>INDUCTION</u> <u>STEP</u>

Hypothesis $\delta'^*([p], \omega) = [\delta^*(p, \omega)]$

Want to show $\forall a \in \Sigma$, $\delta'^*([p]\, \omega a) = [\delta^*(p, \omega a)]$.

Calculate as follows:

$\delta'^*([p], \omega a) = \delta'(\delta'^*([p], \omega), a)$ [Def of δ'^*]

$= \delta'([\delta^*(p, \omega)], a)$ [Ind. hyp]

$= [\delta(\delta^*(p, \omega), a)]$ [Def of δ^*]

$= [\delta^*(p, \omega a)]$ Done.

<u>Thm</u> $L(M') = L(M)$

<u>Proof</u> $x \in L(M') \Leftrightarrow \delta'^*([s_0], x) \in F'$

$\Leftrightarrow [\delta^*(s_0, x)] \in F'$

$\Leftrightarrow \delta^*(s_0, x) \in F$

$\Leftrightarrow x \in L(M)$. ∎

Thus the "collapsed" machine recognizes the same language as the original machine and it has fewer states. Later (Myhill-Nerode) we will see that this is the best possible machine.

Our next task, design an algorithm to minimize DFA. The basic idea is called "splitting": We put all the states into 2 clusters: the accept states & the reject states. Then we keep splitting them by looking at the transitions

Write $p \bowtie q$ if $\exists\, \omega \in \Sigma^*$ s.t.
$\delta^*(p,\omega) \in F$ & $\delta^*(q,\omega) \notin F$
OR $\delta^*(p,\omega) \notin F$ & $\delta^*(q,\omega) \in F$.
We say "p" and "q" are distinguishable.

FACT If $\exists\, a \in \Sigma$ s.t. $\delta(p,a) \bowtie \delta(q,a)$ then $p \bowtie q$.

ALGORITHM Define an $S \times S$ array of booleans.

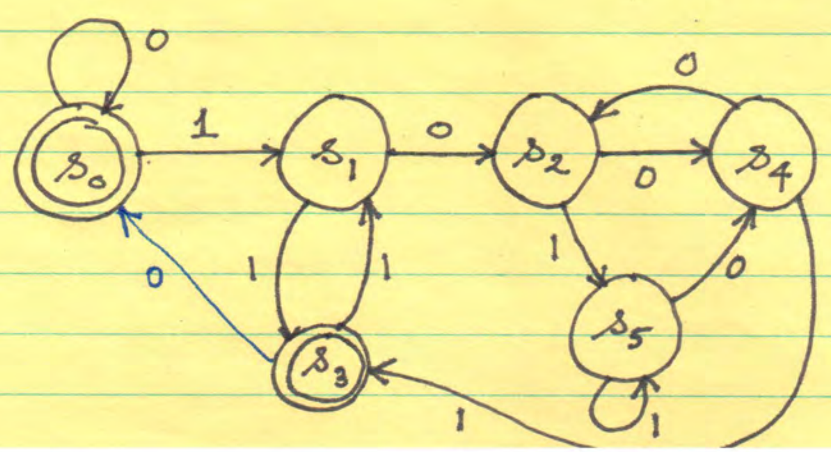
1. For every pair (p,q) s.t. $p \in F$ & $q \notin F$ put a O in the (p,q) cell of the matrix.
2. Repeat until no more changes:
   For each pair (p,q) that are not marked O
   check if $\exists\, a \in \Sigma$ s.t. $(\delta(p,a), \delta(q,a))$ is marked O.
   If yes then mark (p,q) with a O.
3. Mark [illegible]

	s_0	s_1	s_2	s_3	s_4	s_5
s_0						
s_1	0					
s_2	0	0				
s_3	1	0	0			
s_4	0	1	0	0		
s_5	0	0	1	0	0	

DON'T NEED THIS PART OF THE ARRAY.

BLUE : INITIAL STEP
RED : NEXT STEP
GREEN : EQUIVALENT

Terminates in 2 phases.



$s_1 \xrightarrow{1} s_3$, $s_2 \xrightarrow{1} s_4$: $s_3 \times s_4$

$s_2 \xrightarrow{1} s_5$, $s_4 \xrightarrow{1} s_3$: $s_5 \times s_3$

$s_5 \xrightarrow{1} s_5$, $s_1 \xrightarrow{1} s_3$: $s_5 \times s_3$

$s_4 \xrightarrow{1} s_3$, $s_5 \xrightarrow{1} s_5$: $s_3 \times s_5$

Thm If two states are not labelled 0 by the algorithm they are equivalent.

Proof Suppose the machine is $M = (S, s_0, \delta, F)$.
Assume the theorem is false so there is a pair of states (s, t) such that $s \not\equiv t$ but the alg. does not label them. We call this a BAD PAIR.
Among all bad pairs choose ~~the~~ one with the shortest distinguishing string $x = x_1 \dots x_n$; $x_i \in \Sigma$.
So $\delta^*(s, x) \in F$ & $\delta^*(t, x) \notin F$. Note x cannot be empty [why not?].
Now consider $\delta(s, x_1)$ & $\delta(t, x_1)$. This pair of states is not equivalent since

$\delta^*(\delta(s, x_1), x_2 \dots x_n) \in F$
& $\delta^*(\delta(t, x_1), x_2 \dots x_n) \notin F$.

This cannot be a bad pair since their distinguishing string is shorter than x. So the algorithm must have marked $(\delta(s, x_1), \delta(t, x_1))$ at some stage. But then at the next stage it will mark (s, t) with 0. ⊗
Thus there cannot be any bad pairs. ∎

RUNNING TIME (a) $O(n^2)$ pairs in an n-state machine.
Every round takes $O(n^2)$ so $O(n^4)$.

(b) Improvement: Maintain lists of dependencies.
For each pair (s, t) we maintain a list of pairs that are distinguishable if (s, t) turn out to be distinct.
For each pair (s, t) and each $a \in \Sigma$ we put (s, t) on the list for $(\delta(s, a), \delta(t, a))$. Each pair is on $k = |\Sigma|$ lists.
$O(n^2 k)$.

(c) Hopcroft's algorithm: $O(n \log n)$

(d) Brzozowski's algorithm $O(2^n)$!!