Minimal DFA

Among the many DFAs accepting the same regular language L, there is exactly one (up to renaming of states) which has the smallest possible number of states.

Moreover, it is possible to obtain that *minimal DFA for L* starting from any other by the State Minimization Algorithm.

Deciding Equivalence of DFAs

With this algorithm:

We can check whether two DFAs A and B represent the same regular language.

Just minimize both A and B and see if the obtained automata are isomorphic (equal up to state renaming).

Two algorithms

In this lecture we will see two algorithms for minimizing DFAs.

They both work by partitioning the states of the DFA into equivalence classes.

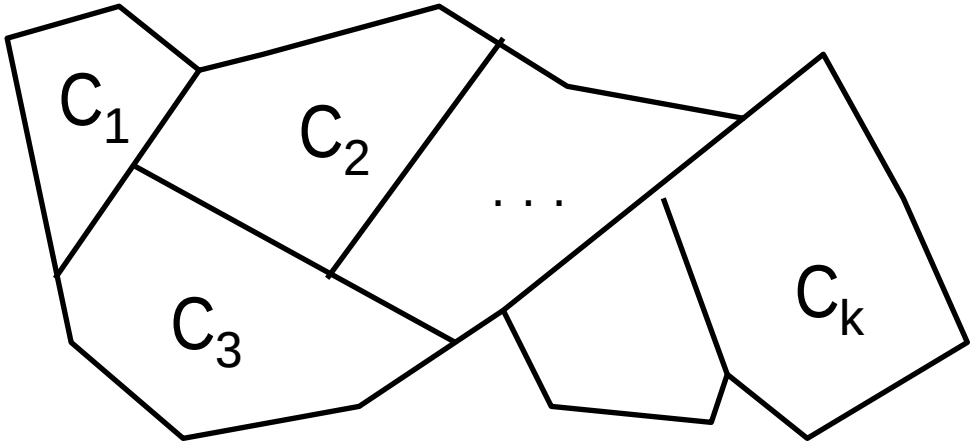
The algorithms differ by how they do the partitioning

- Top Down – From large partitions to smaller ones
- Bottom up – From unit partitions to larger ones

Equivalence Relations and Partitions

A *partition* of a set A is a set $\{C_1, C_2, \ldots, C_k\}$ of subsets of A such that:

- $C_i \neq \emptyset$
- $C_i \cap C_j = \emptyset$ for $i \neq j$
- $C_1 \cup C_2 \cup \ldots \cup C_k = A$



Now if we define a relation ~ on A by: x ~ y **iff** x and y belong to the same partition set; then this relation will satisfy the properties of an *equivalence relation:*

- x ~ x (reflexivity)
- x ~ y $\Rightarrow$ y ~ x (symmetry)
- x ~ y and y ~ z $\Rightarrow$ x ~ z (transitivity)

The correspondence between partitions and equivalence relations goes both ways. If we have an equivalence relation $\approx$ on A then there is an associated partition of the set A. Its classes are called *equivalence classes* of $\approx$. The class containing x is $[x] = \{ y \in A \mid y \approx x\}$.

Equivalence of States

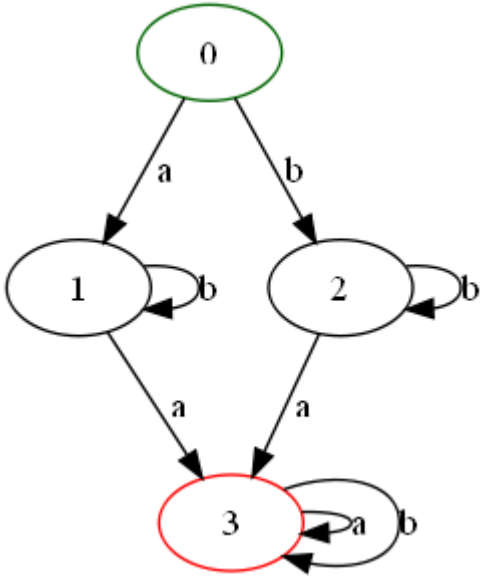
Definition. Two states p and q of a DFA are *equivalent* (notation: $p \equiv q$) when, for every string w: w leads from p to a final state if and only if w leads from q to a final state.

Non-equivalent states are called *distinguishable*. To show that two states are distinguishable, we need to find only one string that leads from one of them to a final state, and leads from the other to a non-final state.

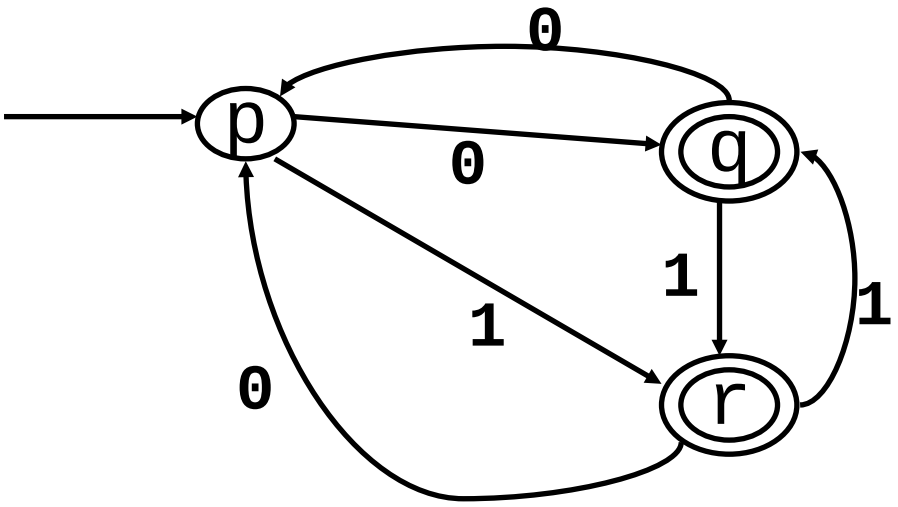
The empty string distinguishes every final state from every non-final state. Thus, if $p \equiv q$, then p and q are either both final or both non-final.

Example

- States 0 and 1 are distinguishable, since
  - “a” from 0 -> 1 Non-final
  - “a” from 1 -> 3, Final



- States 1 and 2 are indistinguishable by any string over the alphabet {a,b}
  - All strings that lead from 1 or 2, that end up in a final state from 1, also lead to a final state from 2!



Testing two final (or two non-final) states for equivalence is more difficult. In this example, `q` and `r` are equivalent. No string beginning with `0` can distinguish them: the two paths converge after the first arc. For strings beginning with `1`, the two paths will alternate between `q` and `r` while reading the initial `1`'s; as soon as the first `0` is read (if ever), the paths will converge at `p`.

Minimal Automata

Definition. A DFA is *minimal* iff

1. All its states are reachable from the start state;
2. All its states are distinguishable.

Theorem. Two minimal automata for the same language are isomorphic.

The quotient

The Quotient (State Collapse) Construction.

State equivalence relation has this fundamental property:
If $p \equiv q$ then for every $a \in \Sigma$, $\delta(p,a) \equiv \delta(q,a)$.

Indeed, if a string w distinguishes $\delta(p,a)$ from $\delta(q,a)$, then the string aw will distinguish p from q.

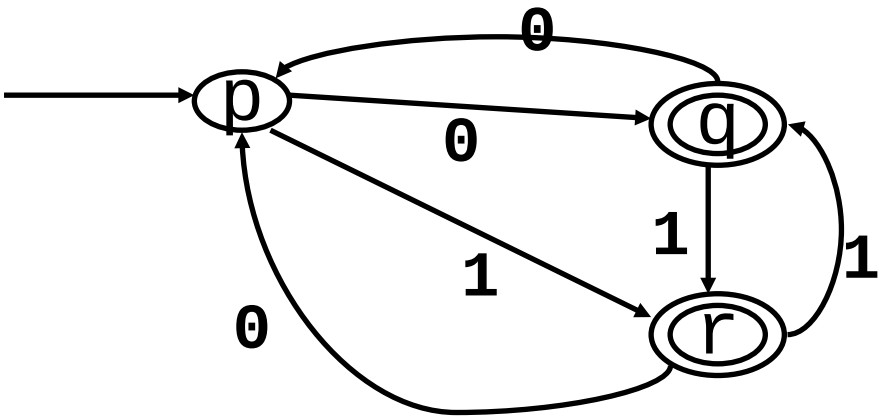
The quotient construction

Given any DFA A

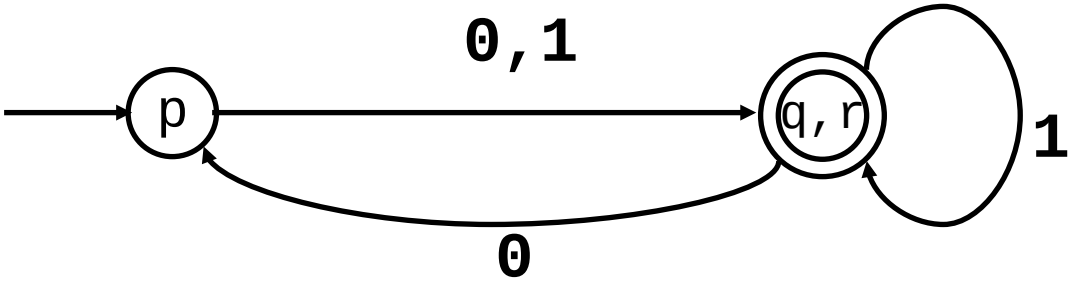
We can construct the *quotient DFA* $A_{\equiv}$

1. Its states are the equivalence classes of the states of A
2. Its transition relation is $\delta_{\equiv}(C,a) = C'$ where
   1. $\delta(x,a)=y$
   2. C is the class that contains x
   3. C' is the class that contains y
3. The start state of $A_{\equiv}$ is the equivalence class of the start state of A,
4. The final states of $A_{\equiv}$ are equivalence classes of final states of A.

Quotient Example



Here is the quotient of the DFA of our example.



Theorem. For any DFA, A, in which all states are reachable from the start state, the automaton $A_{\equiv}$ is minimal.

Minimization Algorithm

Any minimization algorithm has two parts:

- Eliminate states of the input DFA that are not reachable from the start state.
- Do the quotient construction on the resulting DFA.

The first step is easy. The second is easy once we manage to partition the states into equivalence classses.

There are two possible ways to do this

1. Top down - equivalence classes of $\equiv$ are found by an iterative construction that produces a sequence of finer and finer partitions.

2. Bottom up – we construct small (size 2) partitions and then merge them (Algorithm in the Hein Text book)

Top Down approach

Define $p \equiv_i q$ to mean "p and q cannot be distinguished by strings of length $\leq$ i". We can always compute directly $\equiv_0$; it has two classes: *final states* and *non-final states*.

The fact $p \equiv_{i+1} q \Leftrightarrow p \equiv_i q$ and $\delta(p,a) \equiv_i \delta(q,a)$ for every $a \in \Sigma$ is the basis for effectively constructing $\equiv_{i+1}$ when $\equiv_i$ is known.

For each equivalence class, try and break it into two or more new equivalence classes by finding some symbol which distinguishes two states in the current class.

Structure

Base Case

If an equivalence class has only one state, we're done with that state. It cannot be broken down into a "finer" equivalence.

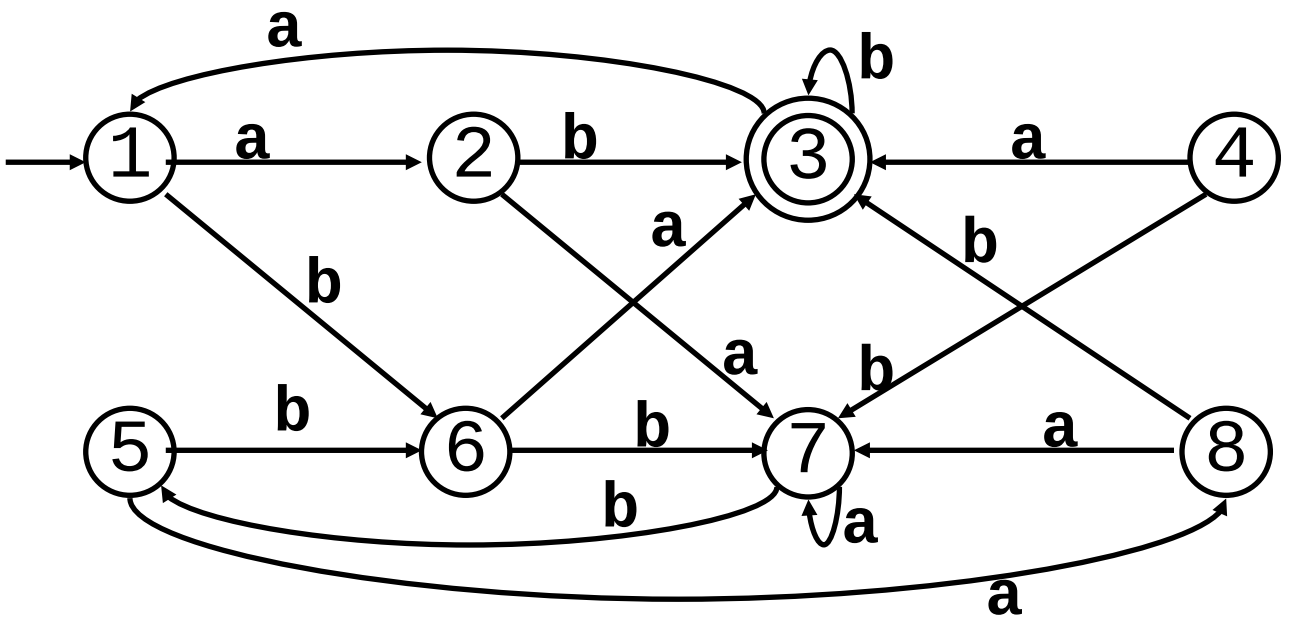
Iterative Step

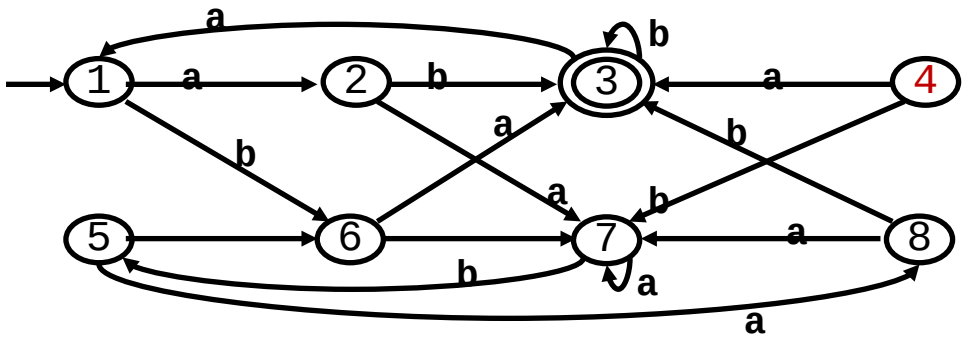
We repeat this construction until $(\equiv_{i+1}) = (\equiv_i)$. Going further on would not change the equivalence relation. At this point we have our desired $\equiv$.

Note the worst we can do is have every state in its own equivalence class, in which case the original DFA was already minimal.

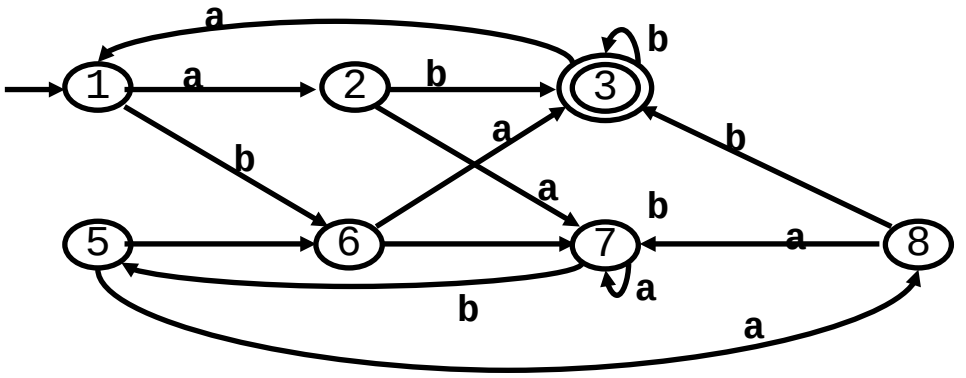
Example

We want the minimal automaton equivalent to the following.



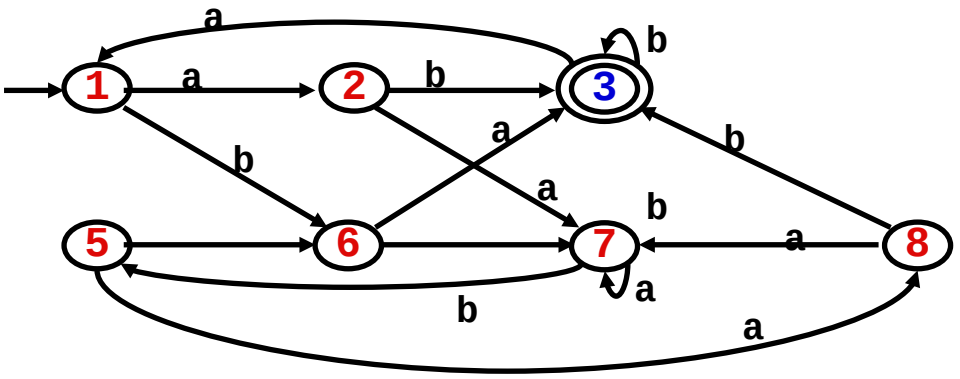


First remove the unreachable state 4



The transition table is now useful.

	1	2	3	5	6	7	8
a	2	7	1	8	3	7	7
b	6	3	3	6	7	5	3



We start with $\equiv_0$; its two classes are X= {1,2,5,6,7} and Y={3}

The equivalence class {3} is done. It cannot be further broken down into finer equivalences. So concentrate on {1,2,5,6,7}

	1	2	3	5	6	7	8
a	2	7	1	8	3	7	7
b	6	3	3	6	7	5	3

Focus on the colors, not the states

	1	2	3	5	6	7	8
a	X	X	X	X	Y	X	X
b	X	Y	Y	X	X	X	Y

Now we need to find $\equiv_1$; We need to look at the table for the states in X={1,2,5,6,7} without distinguishing between states in the same $\equiv_0$ class. We do this by observing the different colors or symbols in each column of the new transition table.

Recall we can ignore the columns labeled by Y= {3} because it is already fully refined (as small as possible).

	1	2	3	5	6	7	8
a	2	7	7	8	3	7	7
b	6	3	3	6	7	5	3

	1	2	3	5	6	7	8
a	X	X	X	X	Y	X	X
b	X	Y	Y	X	X	X	Y

	1	2	3	5	6	7	8
a	2	7		8	3	7	7
b	6	3		6	7	5	3

	1	2	3	5	6	7	8
a	X	X		X	Y	X	X
b	X	Y		X	X	X	Y

Two elements of the same $\equiv_0$ class belong to the same $\equiv_1$ class when their corresponding columns are equal (I.e. have the same color or symbol.) So in this table we have three different patterns X/X , X/Y , and Y/X, and thus we break {1,2,5,7,8} into three different equivalent classes.

X/X {1,5,7} = U

X/Y {2,8} = V

Y/X {6} = W

{3} = Y

Remember to propagate the already done states {3}.

Now identify a new color (or symbol) to each of these sets. And redo the table.

	1	2	3	5	6	7	8
a	2	7	1	8	3	7	7
b	6	3	3	6	7	5	3

	1	2	3	5	6	7	8
a	V	U	U	V	Y	U	U
b	W	Y	Y	W	U	U	Y

	1	2	3	5	6	7	8
a	2	7	1	8	3	7	7
b	6	3	3	6	7	5	3

	1	2	3	5	6	7	8
a	V	U	U	V	Y	U	U
b	W	Y	Y	W	U	U	Y

Ignoring the two done states Y={3} and W={6} (of size 1). We try and further refine U= {1,5,7} and V={2,8} . We note that V={2,8} cannot be further refined, but {1,5} can be distinguished from {7} because the transition on b differs. So naming these two states

{1,5} = T

{7} = Z

We are led to the

final transition diagram,

Because further distinction

Between {1,5}, and {2,8}

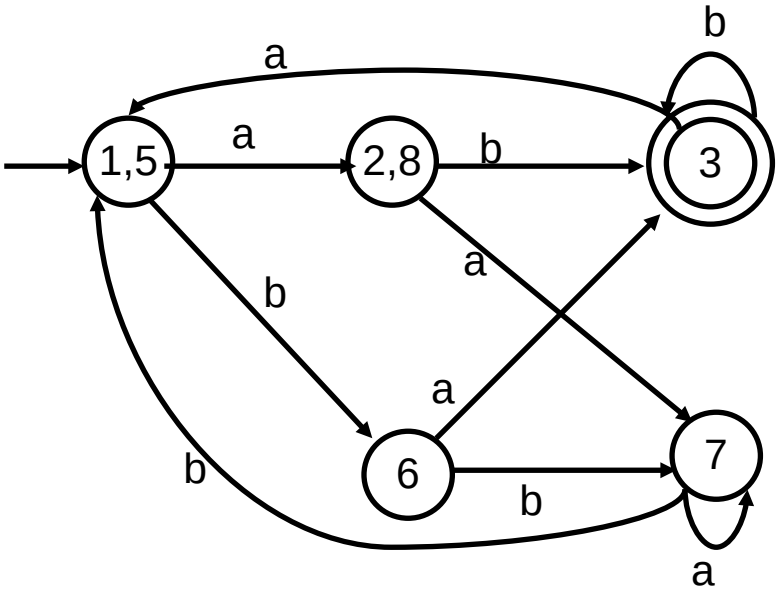
Is not possible.

	1 = T	2 = V	3 = Y	5 = T	6 = W	7 = Z	8 = v
a	V	Z	T	V	Y	Z	Z
b	W	Y	Y	W	Z	T	Y

	1 = T	2 = V	3 = Y	5 = T	6 = W	7 = Z	8 = v
a	V	Z	T	V	Y	Z	Z
b	W	Y	Y	W	Z	T	Y

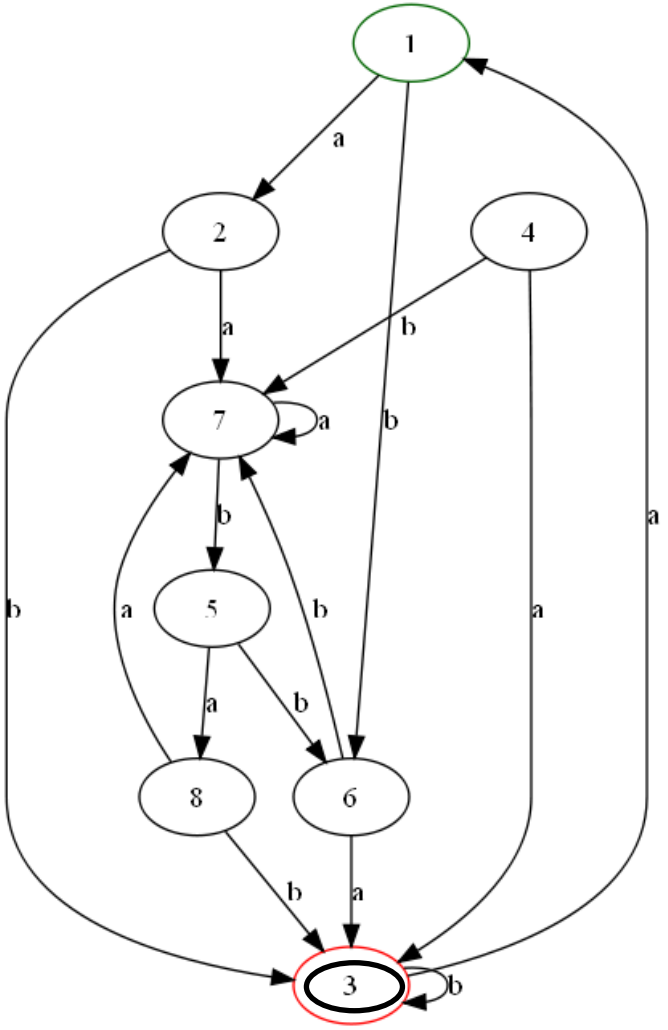
Redundant columns

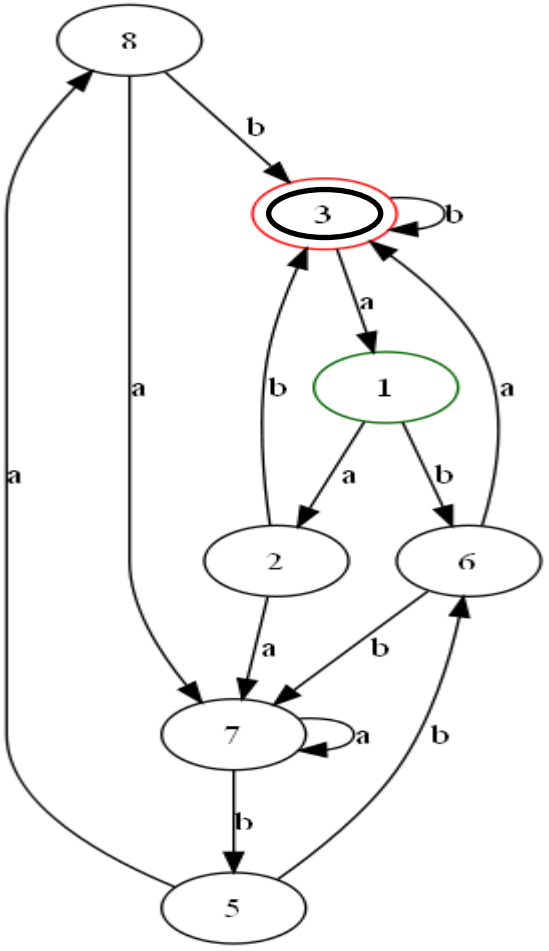
	T={1,5}	V={2,8}	Y={3}	W={6}	Z={7}
a	V={2,8}	Z={7}	T={1,5}	Y={3}	Z={7}
b	W={6}	Y={3}	Y={3}	Z={7}	T={1,5}



Bottom up (in Text Book)

Remove states not reachable from the start state (4)



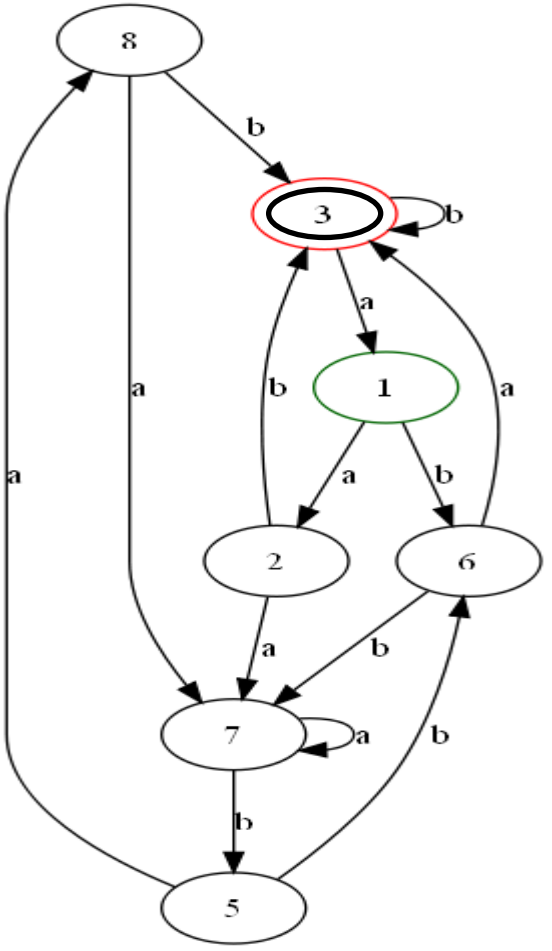


Bottom up, start with small equivalences

Compute all distinct pairs of states
Where both parts of the pair are either
Both final, or both non-final

[(5,8),(5,7),(5,6),(7,8),(6,8),(6,7),
(2,8),(2,5),(2,7),(2,6),(1,8),(1,5),
(1,7),(1,6),(1,2)]

Note that 3 is the only final state, so
It participates in no pair



Keep only good pairs

[(5,8),(5,7), (5,6) , (7,8) ,(6,8),(6,7),(2,8),(2,5),(2,7),
(2,6),(1,8),(1,5),(1,7),(1,6),(1,2)]

→

[(5,7),(2,8),(1,5),(1,7)]

A pair (x,y) is good if

1. Its δ is the same on all letters
   $\delta(x,a) = \delta(y,a)$ for all a

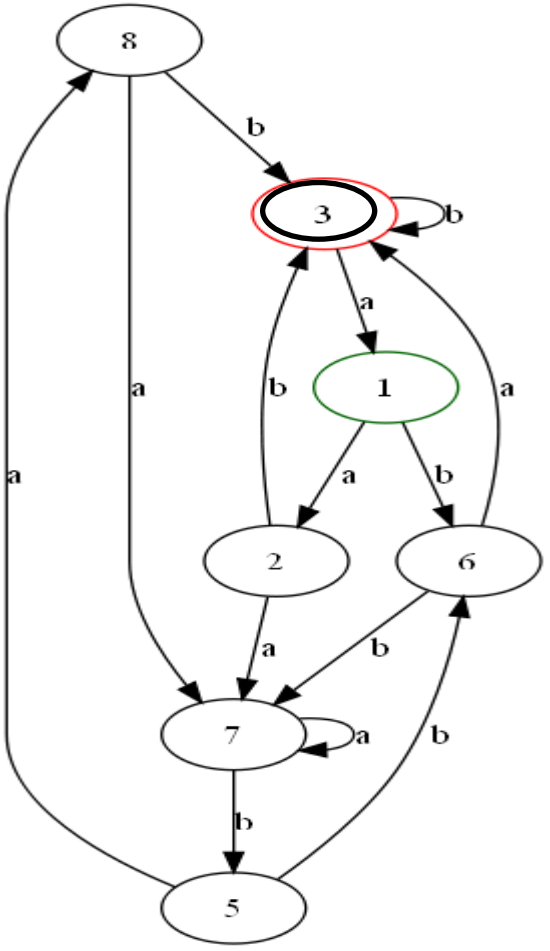
OR

2. The pair ($\delta(x,a)$, $\delta(y,a)$)
   is already in the current set of pairs
   forall a

Example (5,7)

($\delta(5,a)$, $\delta(7,a)$) = (7,8)

($\delta(5,B)$, $\delta(7,B)$) = (6,5)



Repeat

Repeat until no more pairs can be removed

[(5,8),(5,7), (5,6) ,(7,8),(6,8),(6,7),(2,8),(2,5),
(2,7),(2,6),(1,8),(1,5),(1,7),(1,6),(1,2)]

→

[(5,7),(2,8),(1,5),(1,7)]

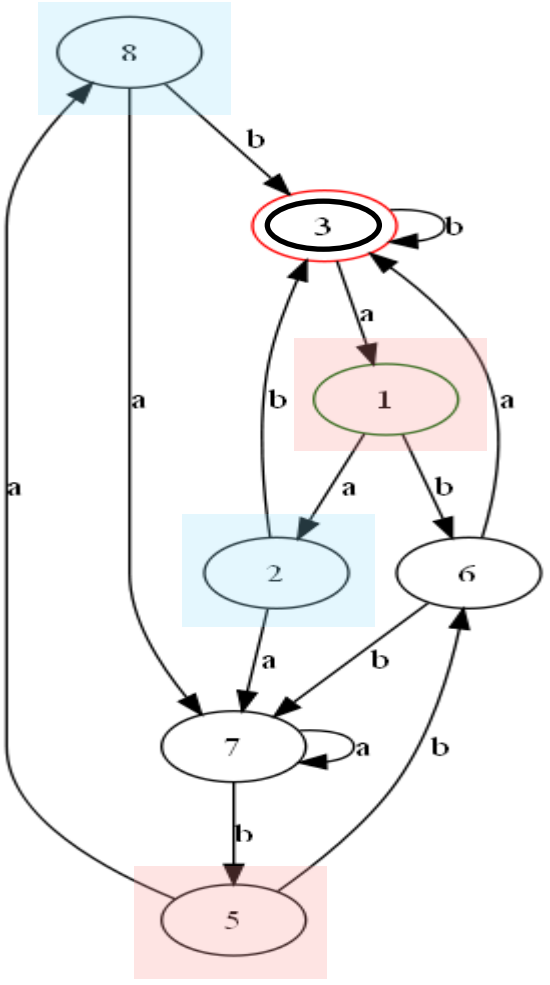
→

[(2,8),(1,5)]

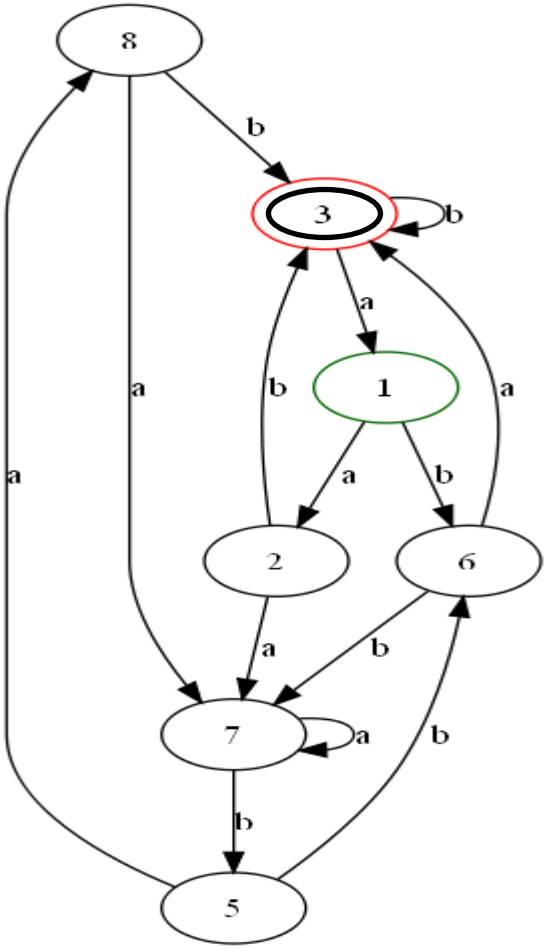
So the equivalences are
{{3},{4},{6},{7},{2,8},{1,5}}

Partition

{{3},{4},{6},{7},{2,8},{1,5}}



New DFA



δ({3},a) = {1,5}
δ({3},b) = {3}
δ({4},a) = ...
δ({4},b) =
δ({6},a) =
δ({6},b) =
δ({7},a) =
δ({7},b) =
δ({2,8},a) =
δ({2,8},b) =
δ({1,5},a) =
δ({1,5},b) =