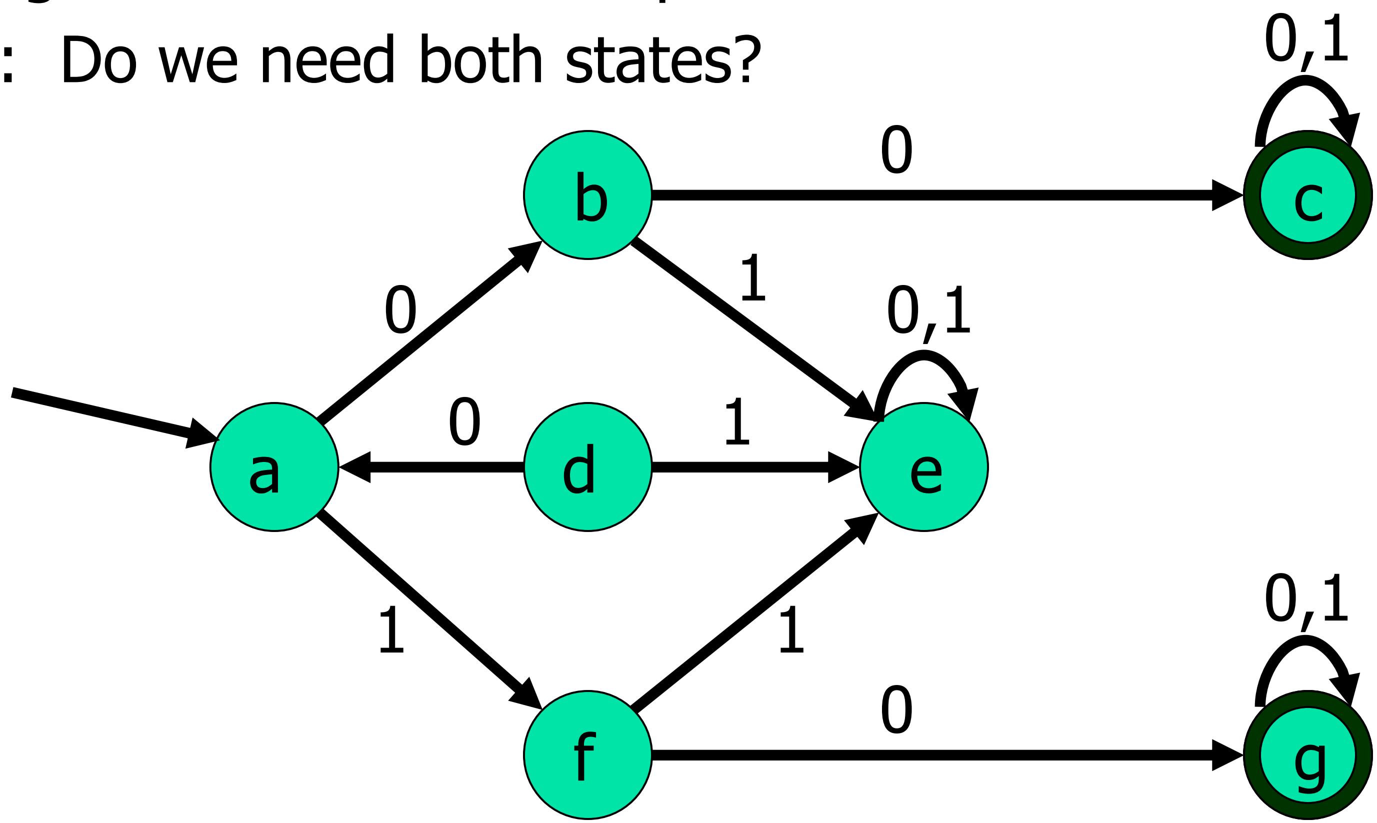
DFA Minimization

Equivalent States

Consider the accept states c and g. They are both sinks meaning that any string which ever reaches them is guaranteed to be accepted later.

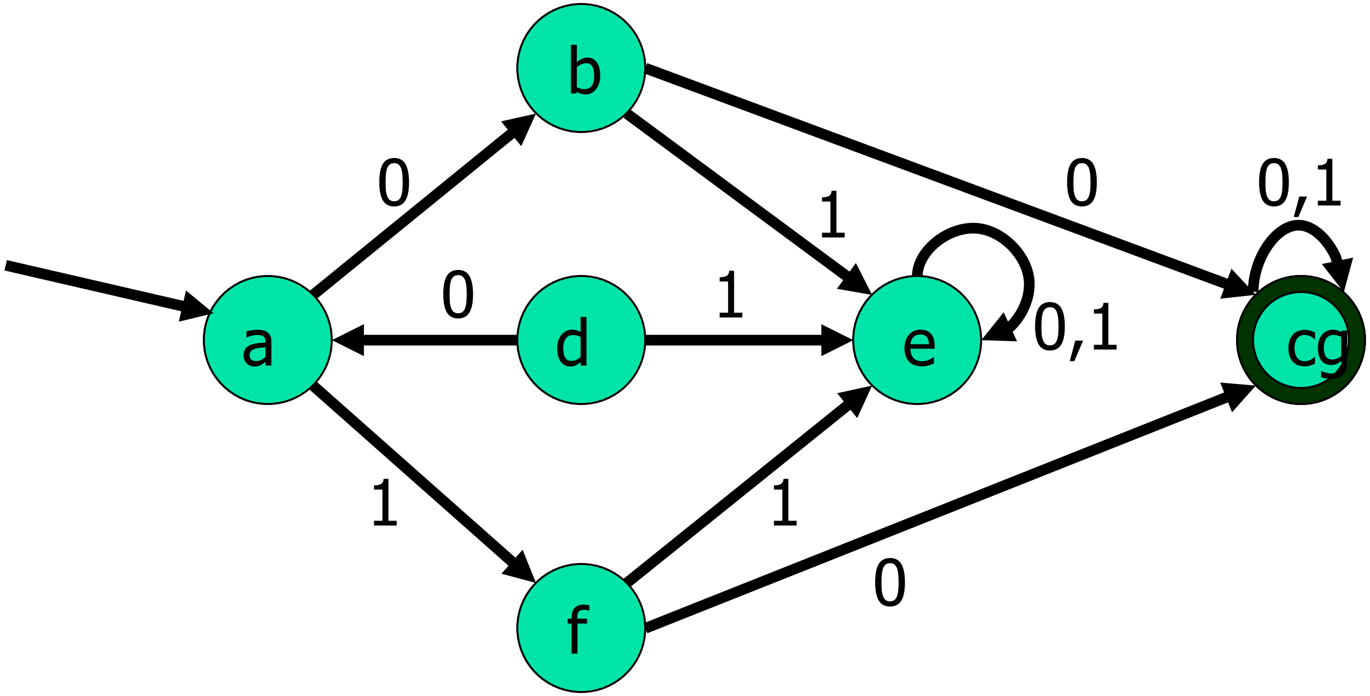
Q: Do we need both states?



Equivalent States

A: No, they can be unified as illustrated below.

Q: Can any other states be unified because any subsequent string suffixes produce identical results?

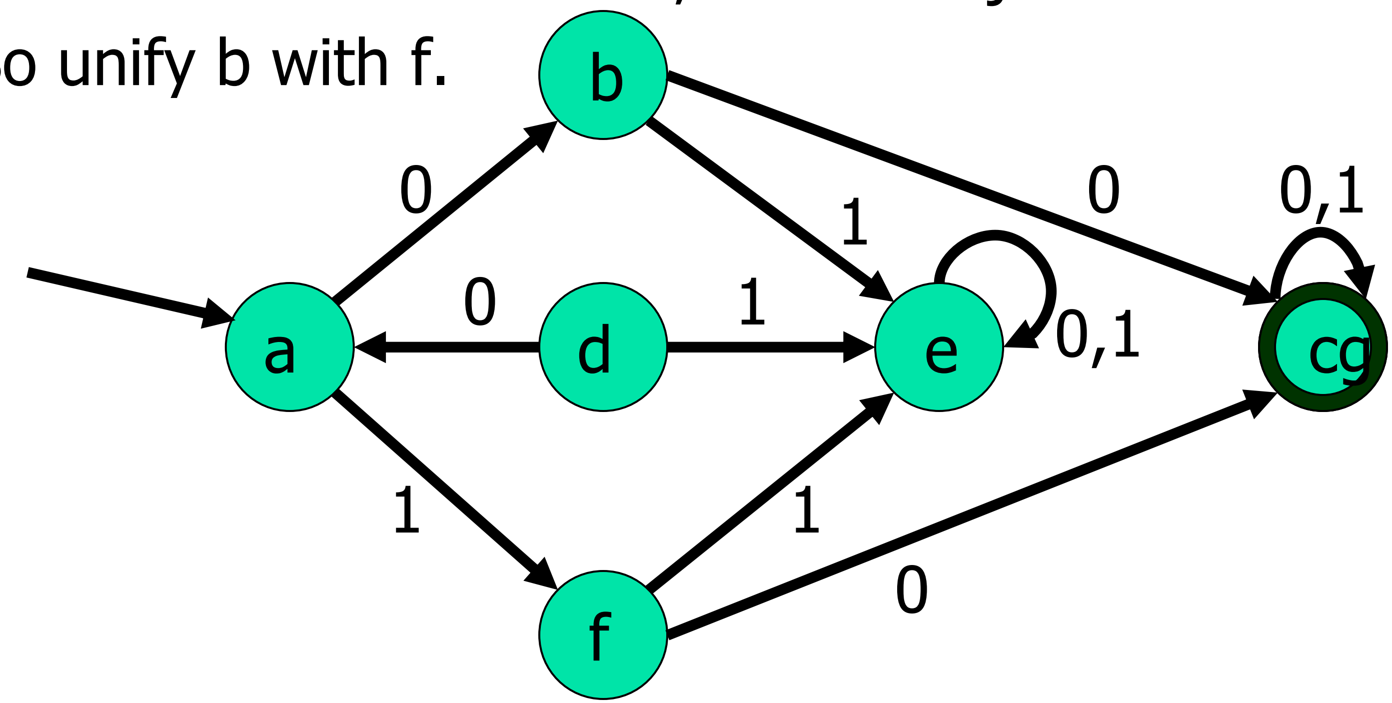


Equivalent States

A: Yes, b and f. Notice that if you're in b or f then:

1. if string ends, reject in both cases
2. if next character is 0, forever accept in both cases
3. if next character is 1, forever reject in both cases

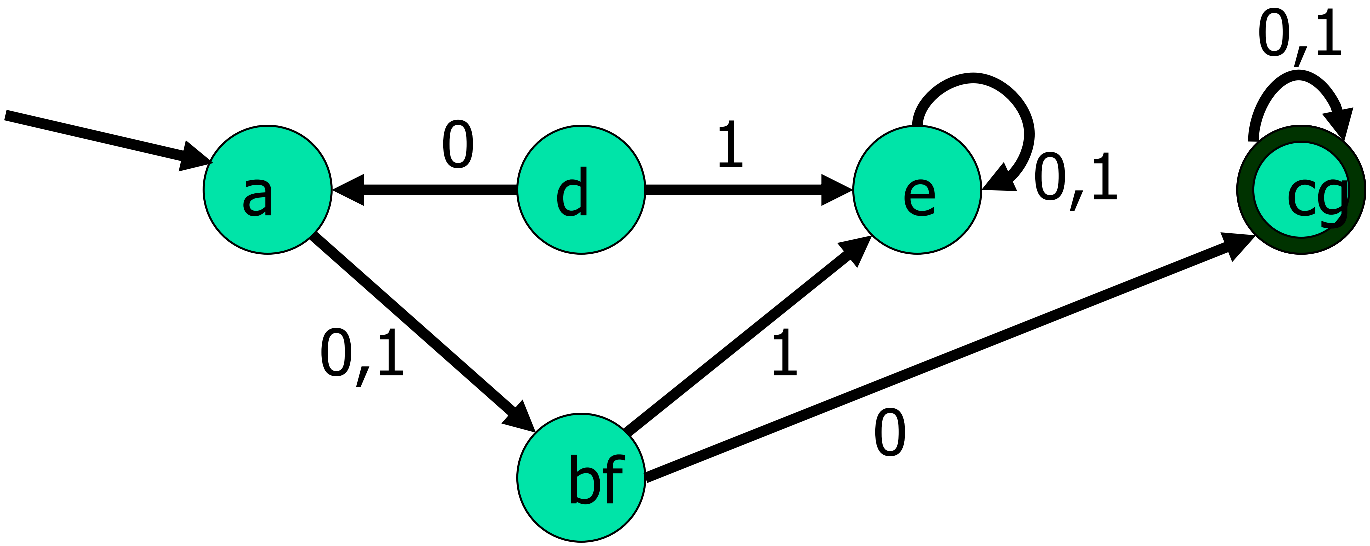
So unify b with f.



Equivalent States

Intuitively two states are equivalent if all subsequent behavior from those states is the same.

Q: Come up with a formal characterization of state equivalence.



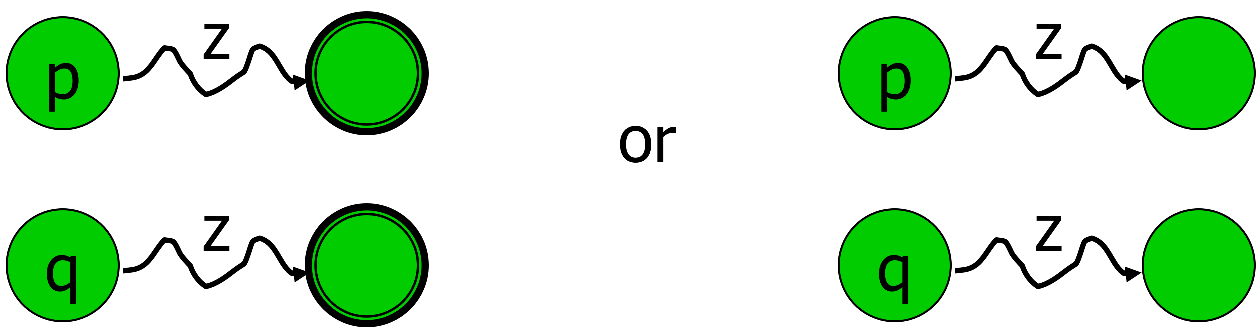
DFA Minimization: Algorithm Idea

Equate & collapse states having same behavior.

Build equivalence relation on states:

$$p \equiv q \quad \leftrightarrow \quad (\forall z \in \Sigma^*,\ \hat{\delta}(p,z) \in F \ \leftrightarrow\ \hat{\delta}(q,z) \in F)$$

I.e., iff for every string z, one of the following is true:



DFA Minimization: Algorithm

Build table to compare each unordered pair of distinct states p,q.

Each table entry has

- a “mark” as to whether p & q are known to be not equivalent, and
- a list of entries, recording dependences: “If this entry is later marked, also mark these.”

DFA Minimization: Algorithm

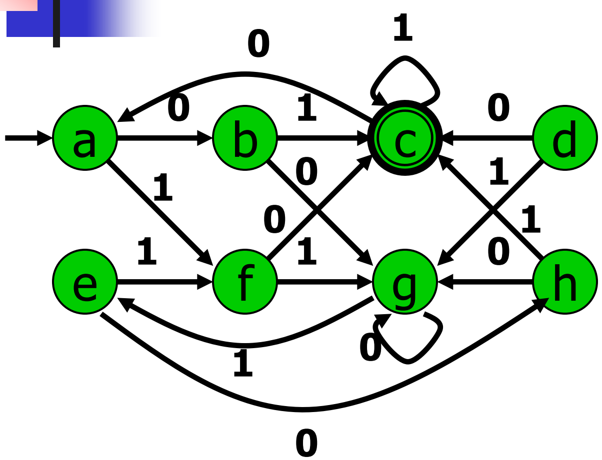
1. Initialize all entries as unmarked & with no dependences.

2. Mark all pairs of a final & nonfinal state.

3. For each unmarked pair p,q & input symbol a:
   1. Let $r=\delta(p,a)$, $s=\delta(q,a)$.
   2. If (r,s) unmarked, add (p,q) to (r,s)'s dependences,
   3. Otherwise mark (p,q), and recursively mark all dependences of newly-marked entries.

4. Coalesce unmarked pairs of states.
5. Delete inaccessible states.

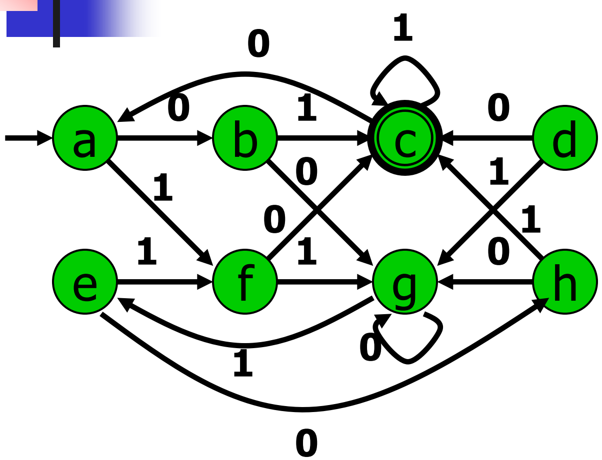
DFA Minimization: Example

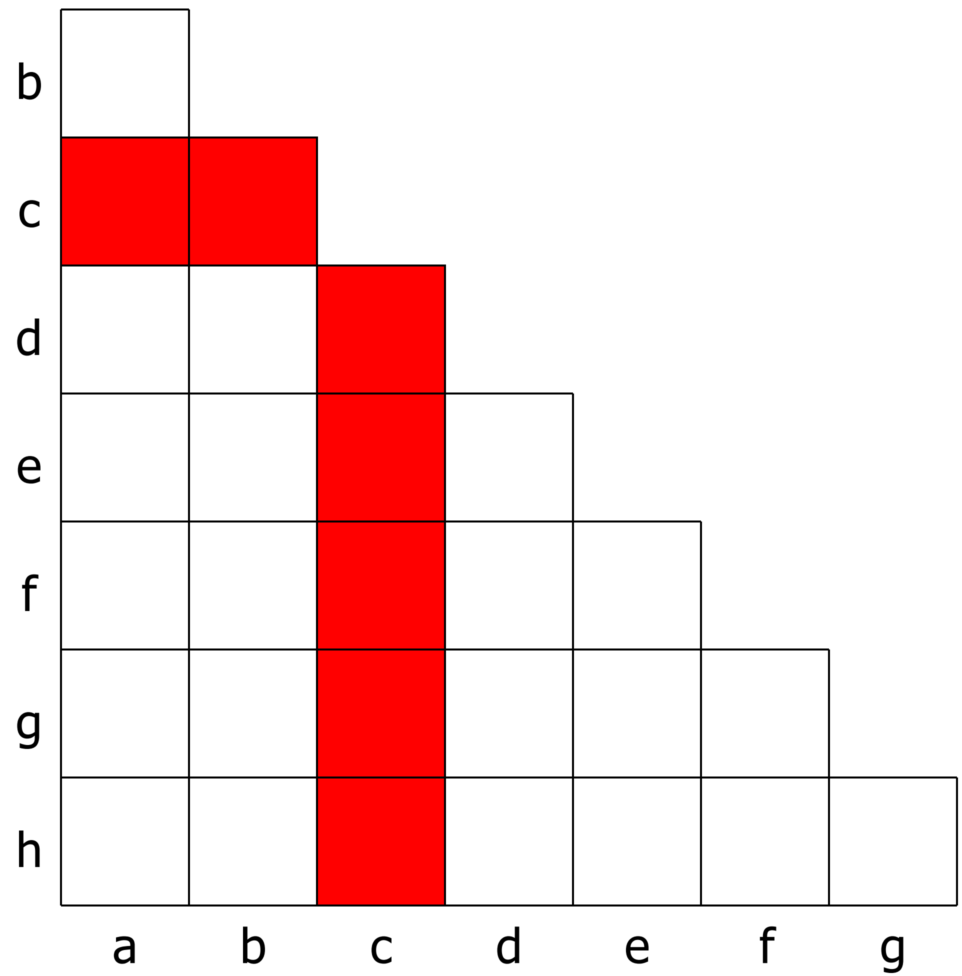


1. Initialize table entries:
   Unmarked, empty list

	a	b	c	d	e	f	g
b							
c							
d							
e							
f							
g							
h							

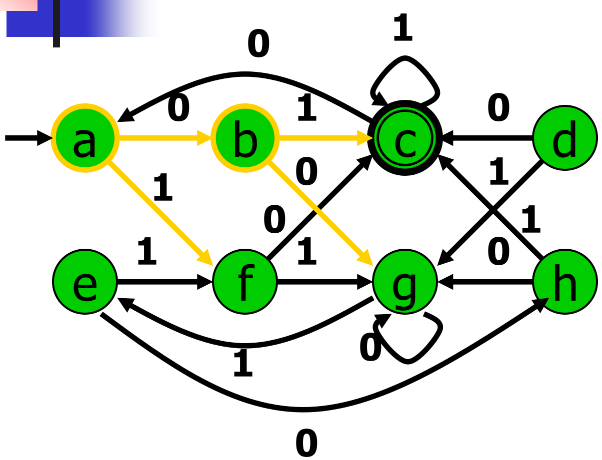
DFA Minimization: Example



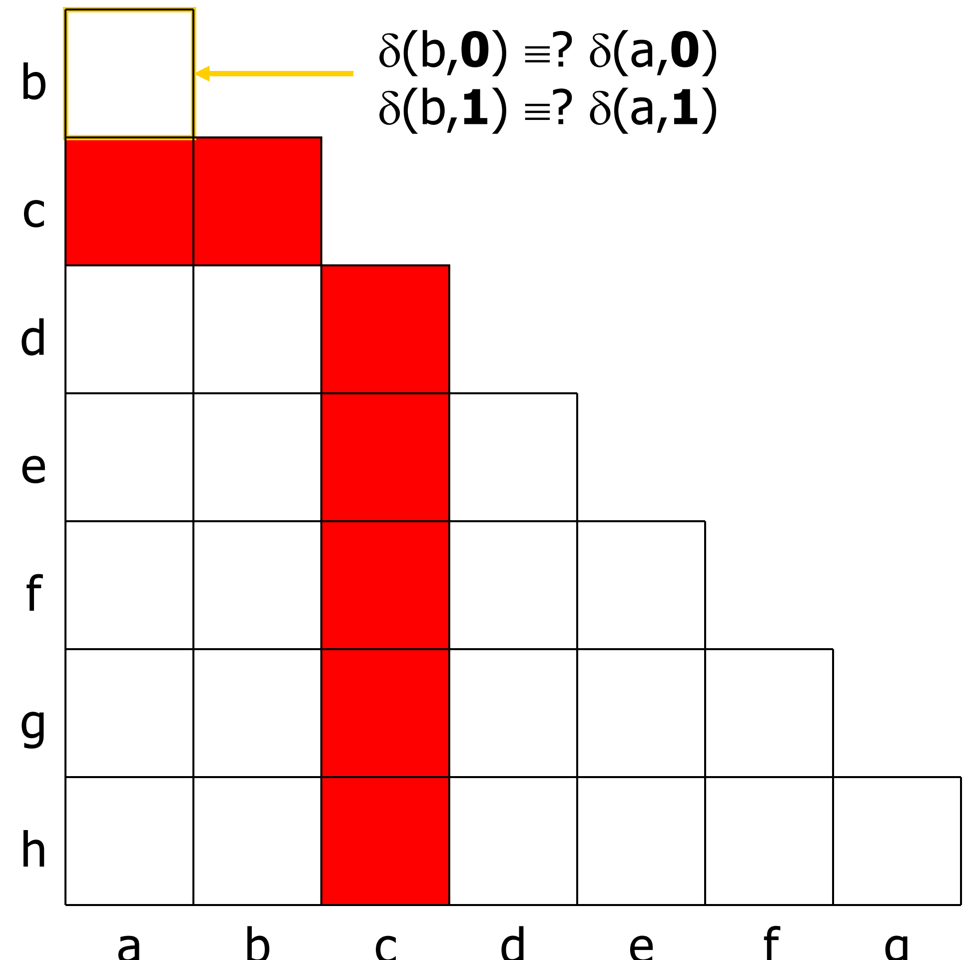


2. Mark pairs of final & nonfinal states

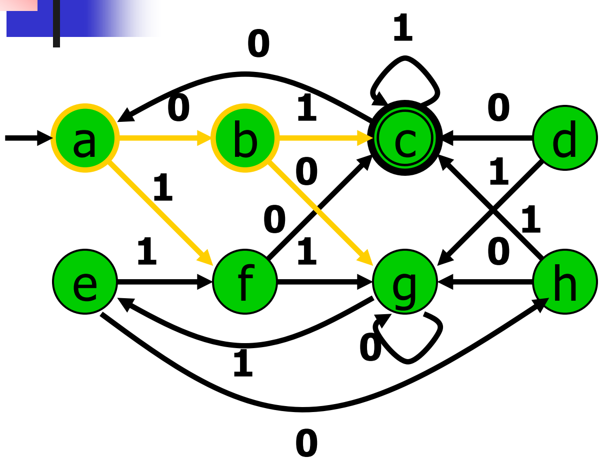
DFA Minimization: Example



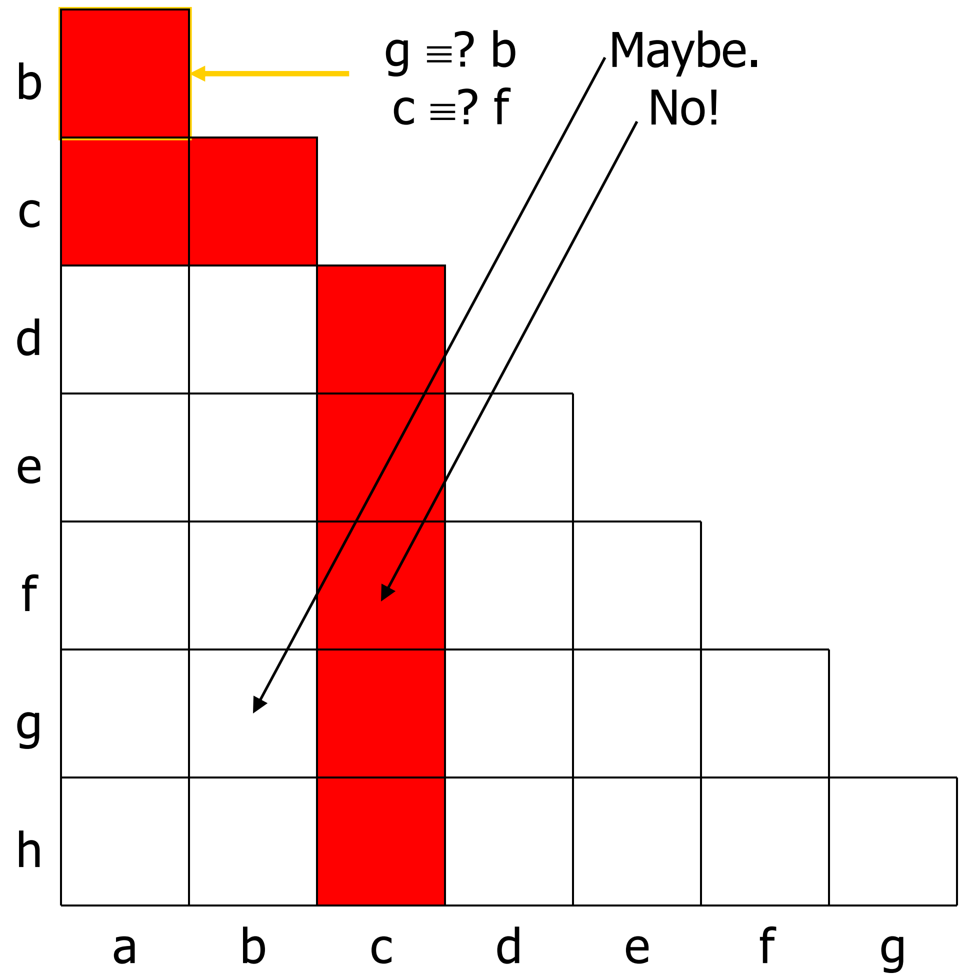
3. For each unmarked pair & symbol, …



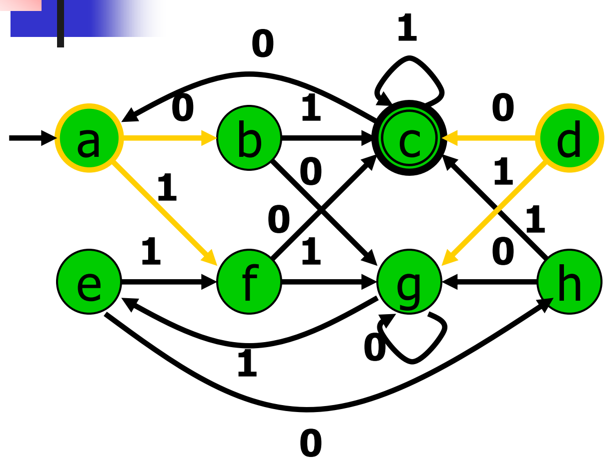
DFA Minimization: Example



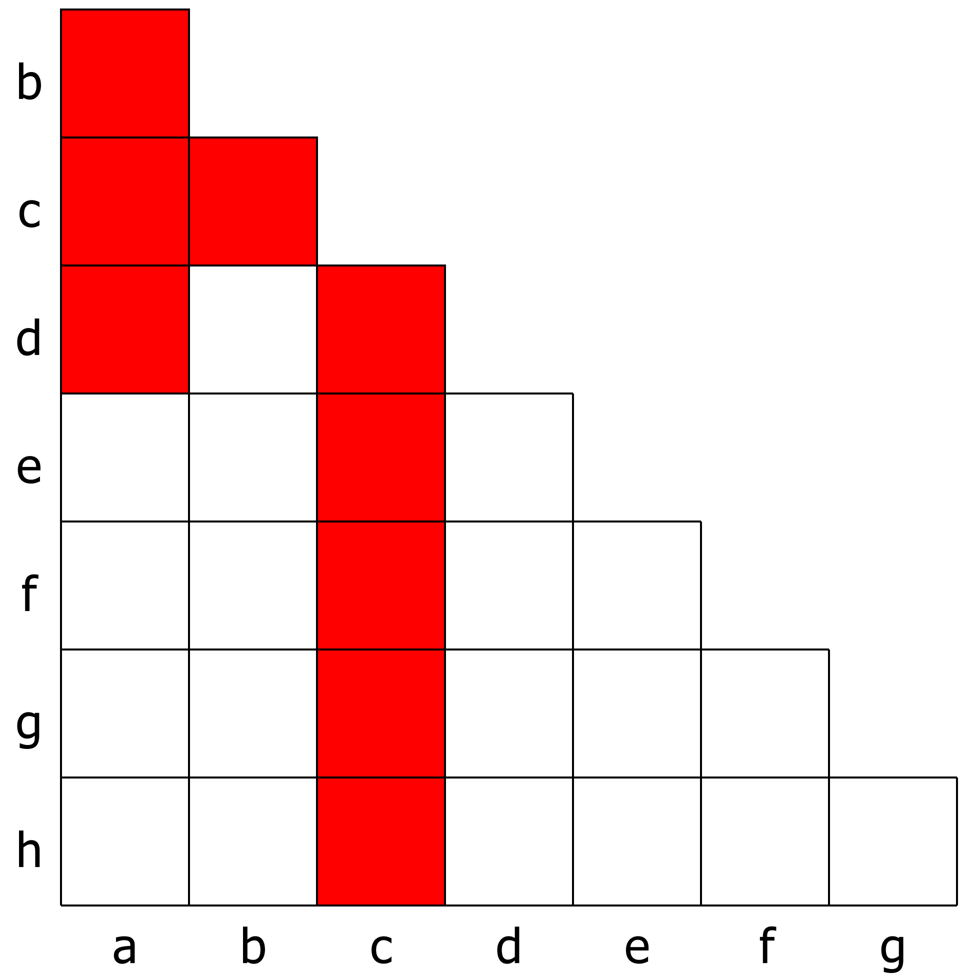
3. For each unmarked pair & symbol, …



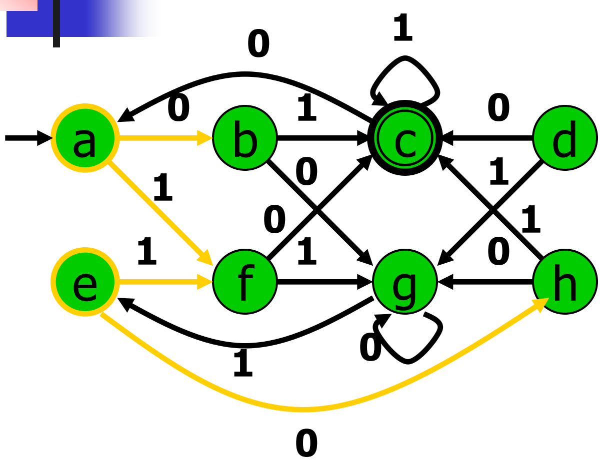
DFA Minimization: Example



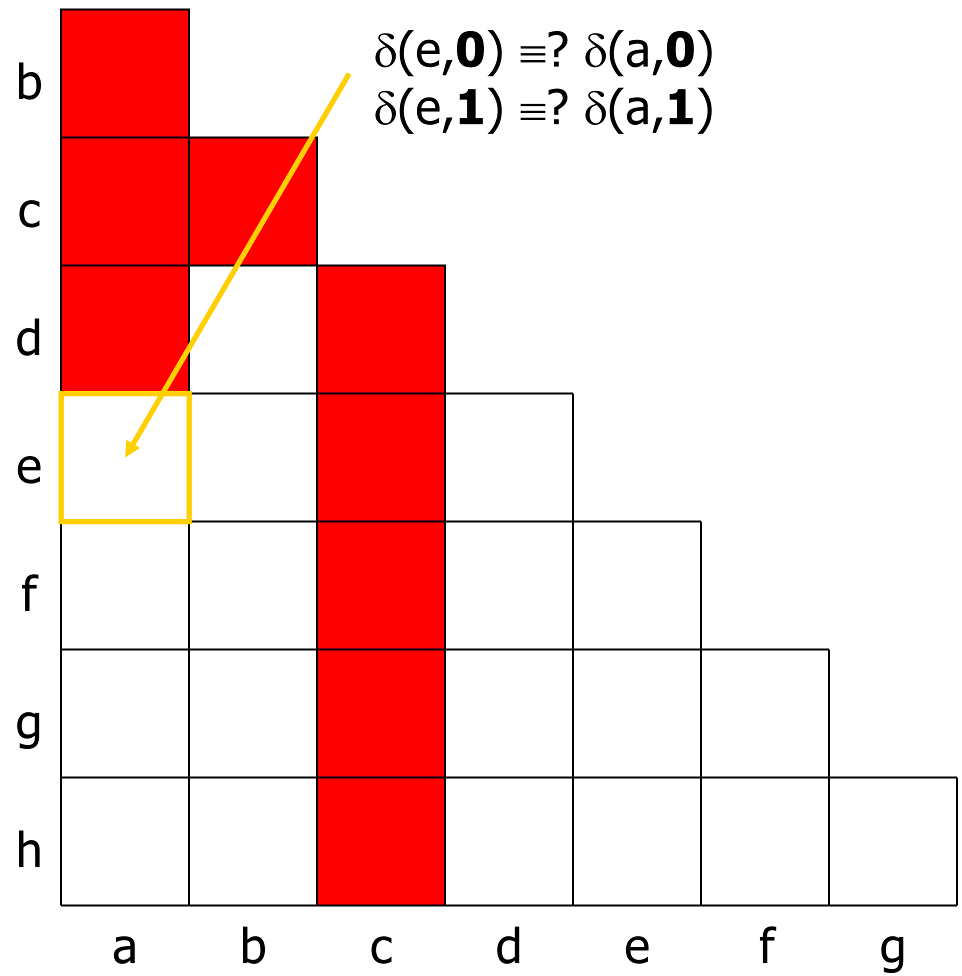
3. For each unmarked pair & symbol, …



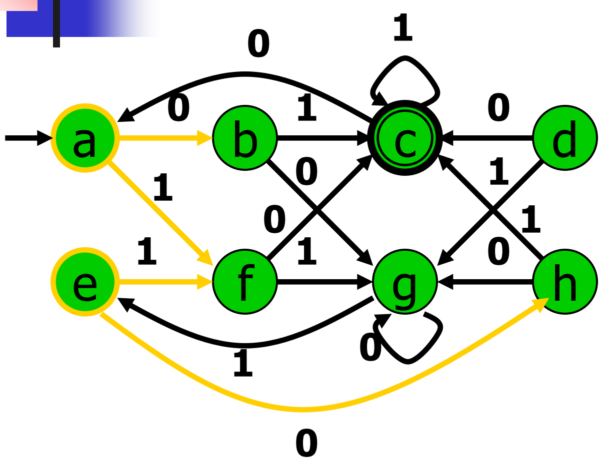
DFA Minimization: Example



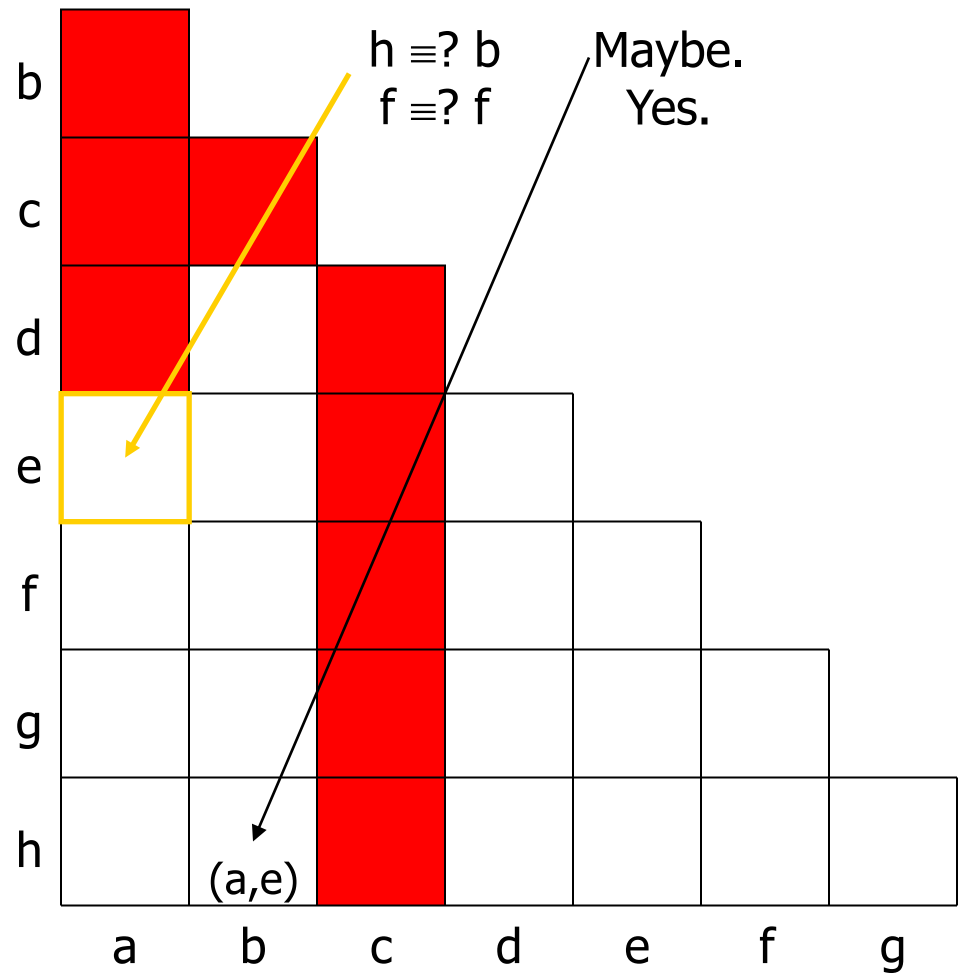
3. For each unmarked pair & symbol, …



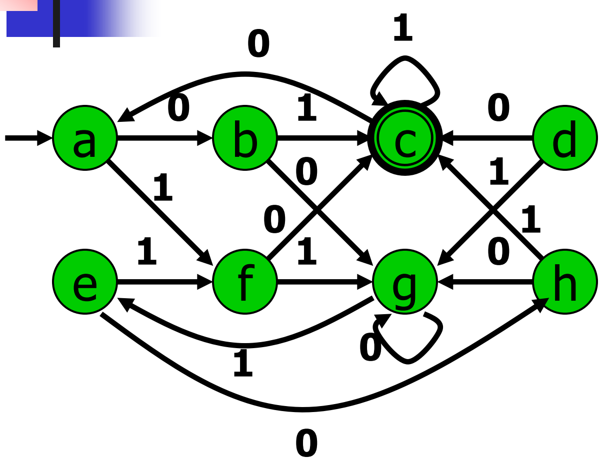
DFA Minimization: Example



3. For each unmarked pair & symbol, …



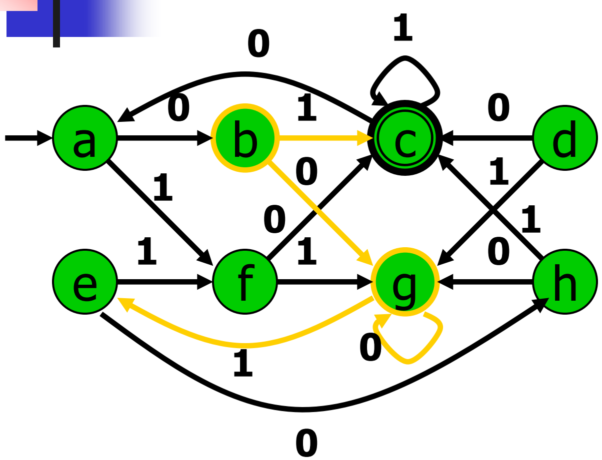
DFA Minimization: Example



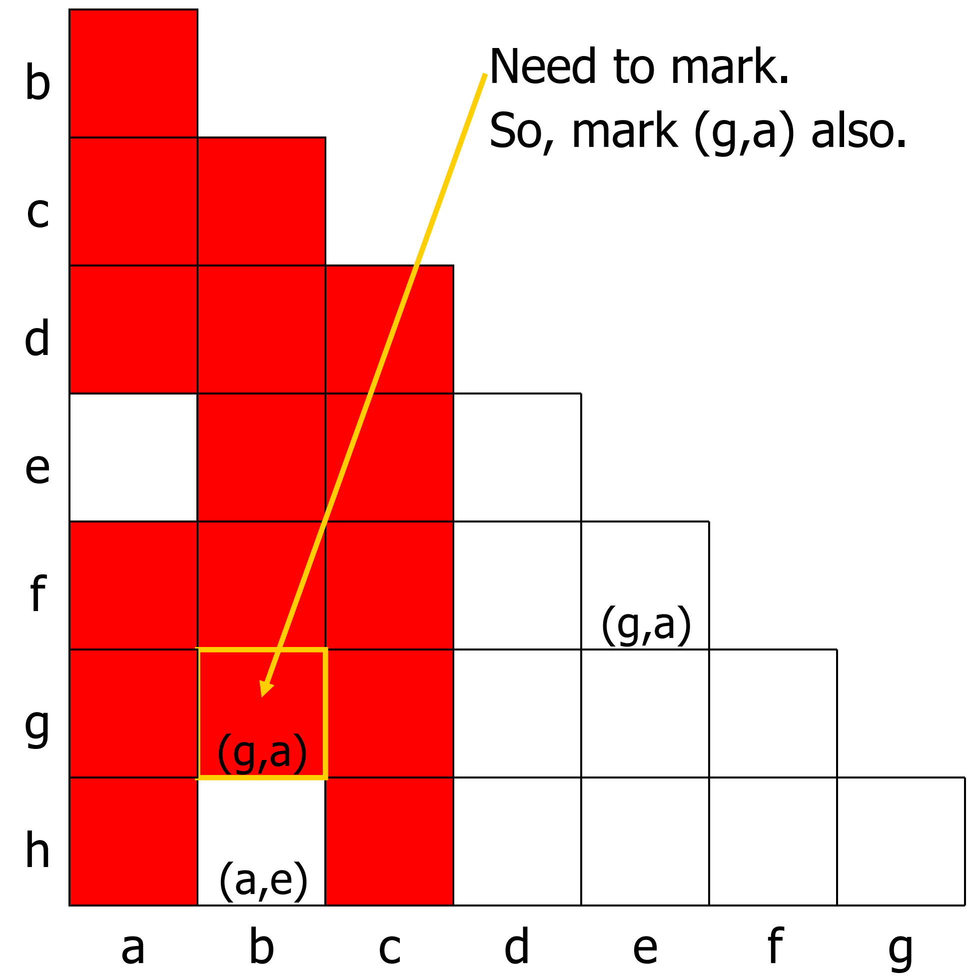
3. For each unmarked pair & symbol, …

	a	b	c	d	e	f	g
b							
c							
d							
e							
f					(g,a)		
g		(g,a)					
h		(a,e)					

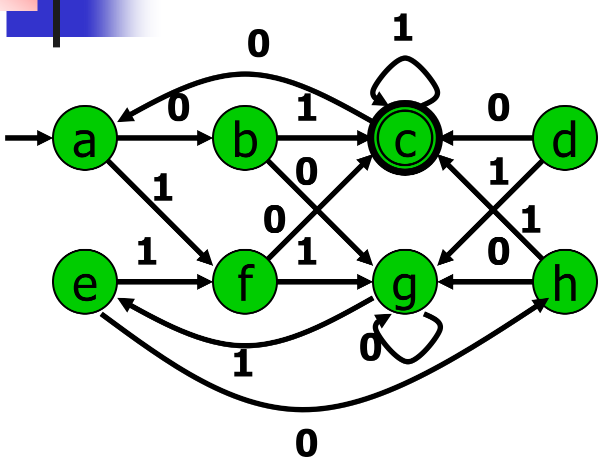
DFA Minimization: Example



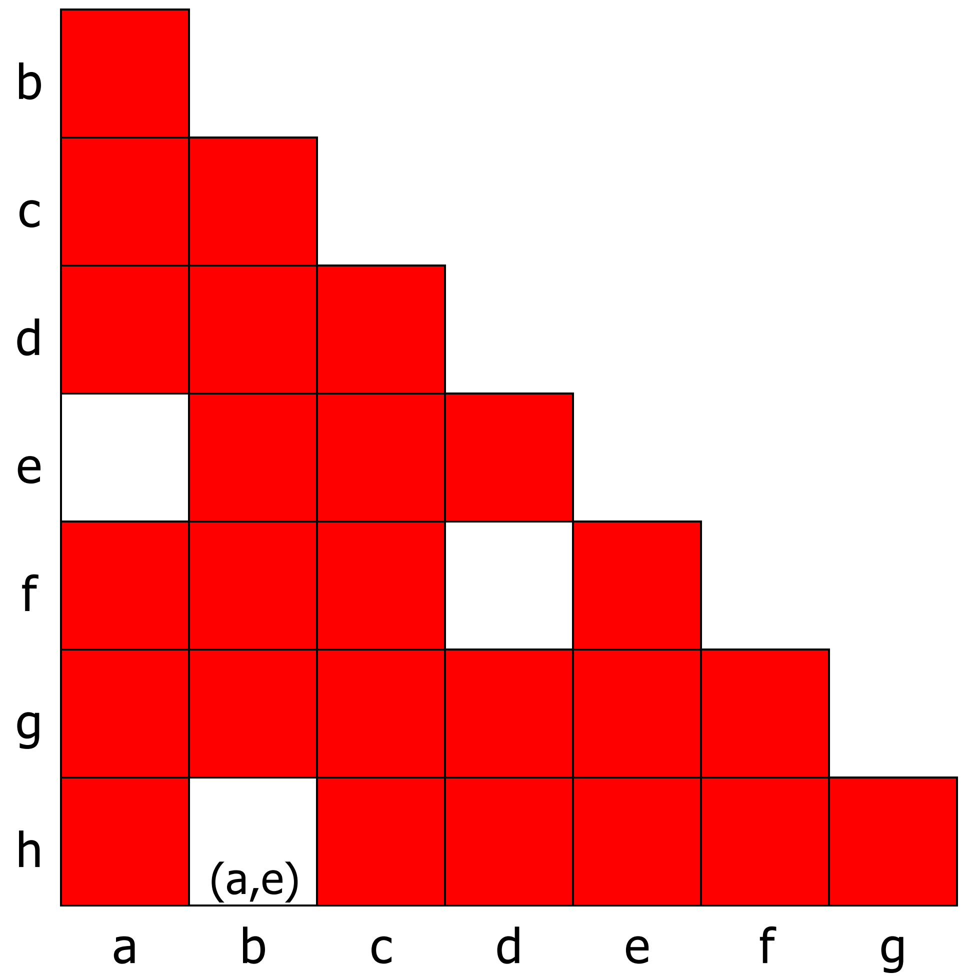
3. For each unmarked pair & symbol, …



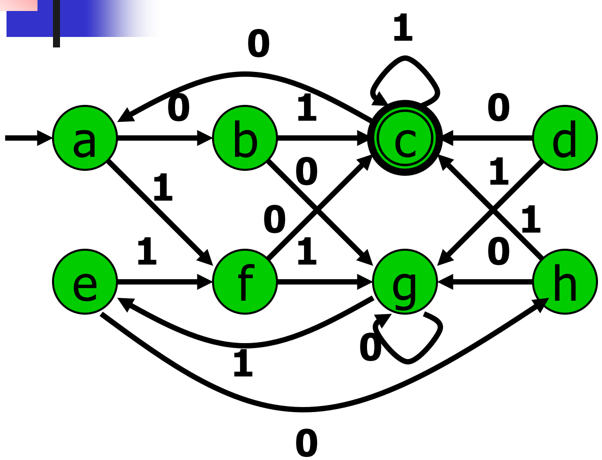
DFA Minimization: Example



3. For each unmarked pair & symbol, …

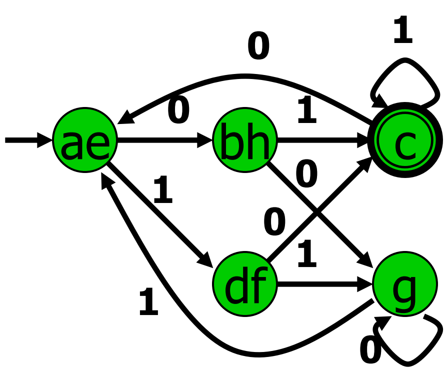


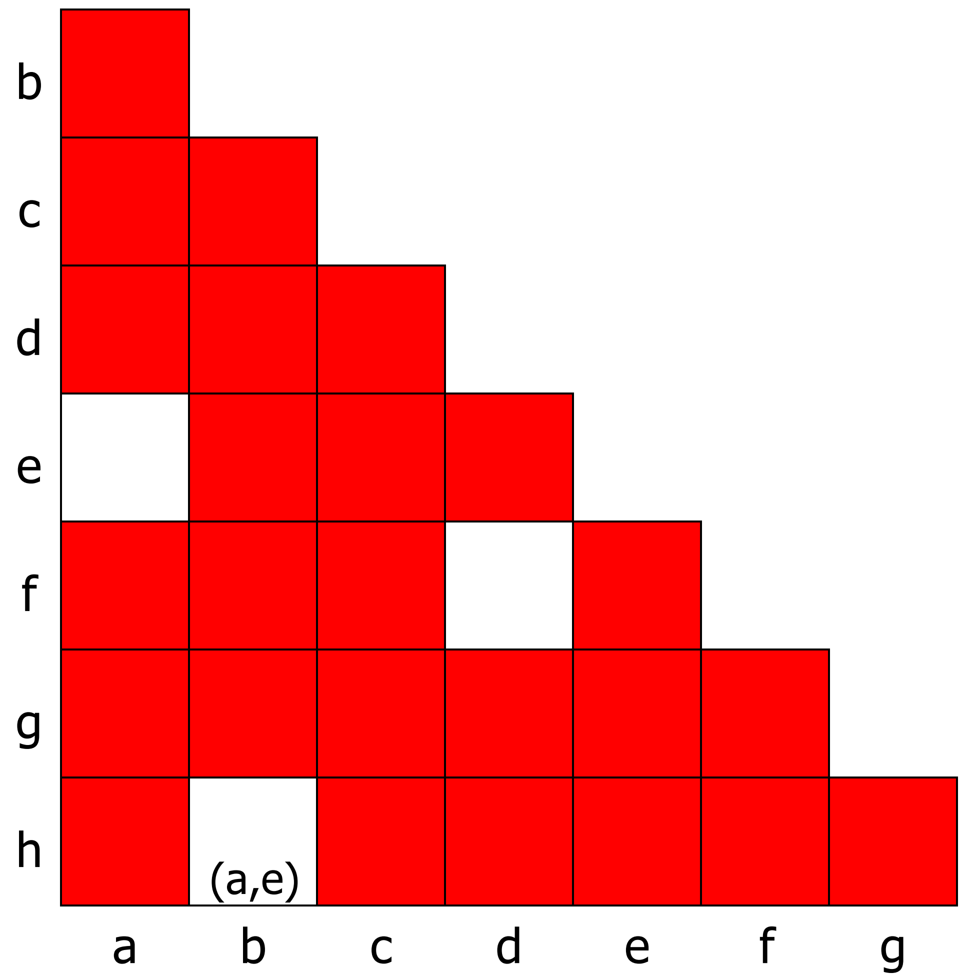
DFA Minimization: Example



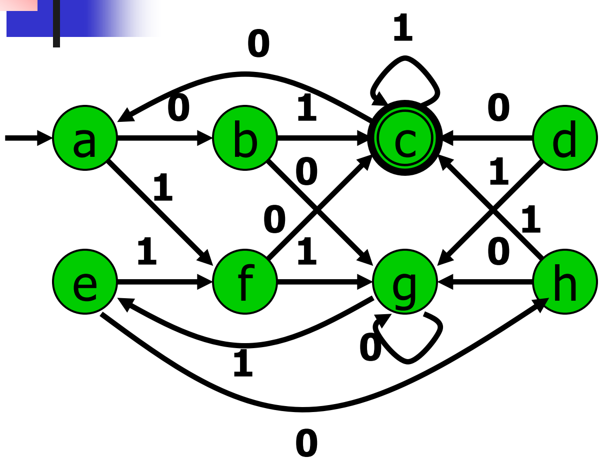
4. Coalesce unmarked pairs of states.

$a \equiv e$
$b \equiv h$
$d \equiv f$



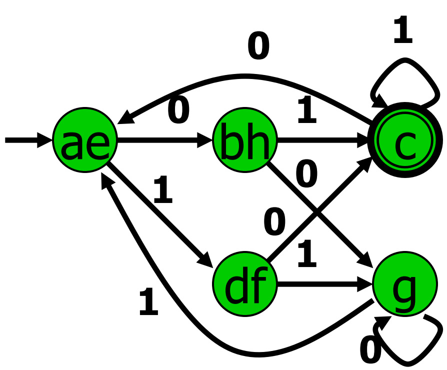


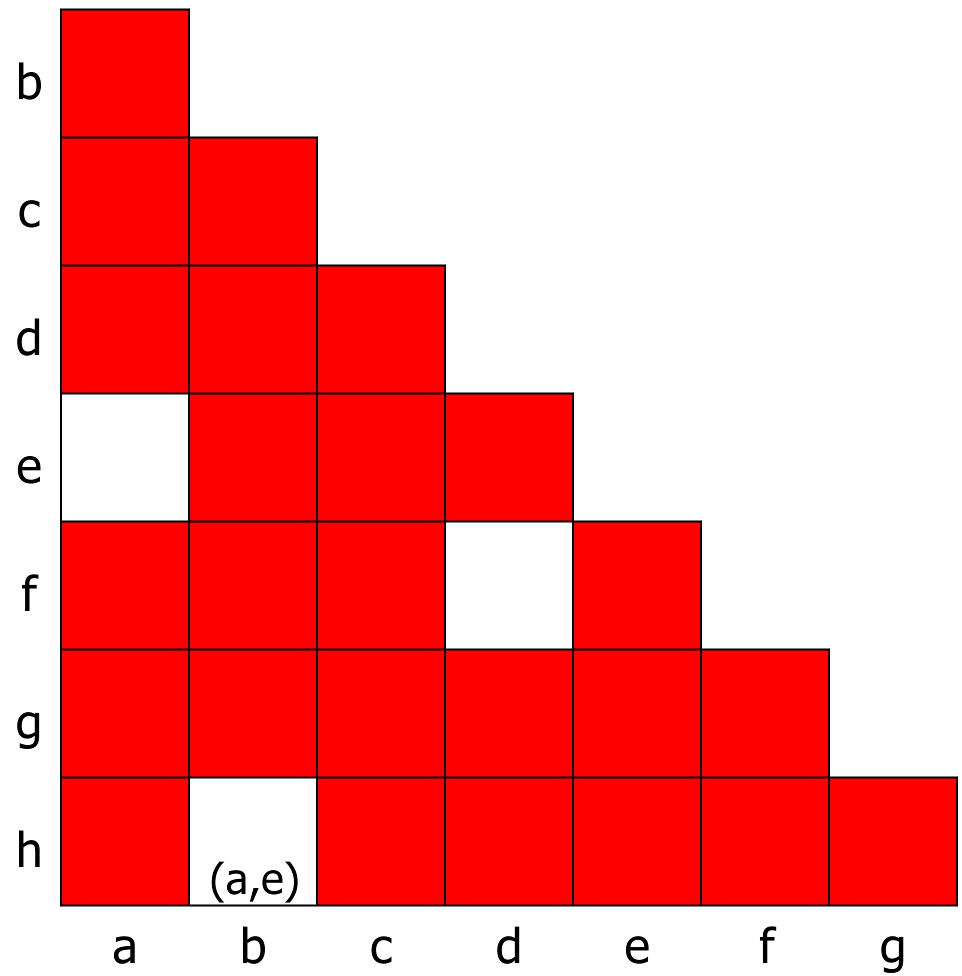
DFA Minimization: Example



5. Delete unreachable states.

None.





DFA Minimization: Notes

Order of selecting state pairs was arbitrary.

- All orders give same ultimate result.
- But, may record more or fewer dependences.
- Choosing states by working backwards from known non-equivalent states produces fewest dependences.

Can delete unreachable states initially, instead.

This algorithm: $O(n^2)$ time; Huffman (1954), Moore (1956).

- Constant work per entry: initial mark test & possibly later chasing of its dependences.
- More efficient algorithms exist, e.g., Hopcroft (1971).

What About NFA Minimization?

This algorithm doesn't find a unique minimal <u>NFA</u>.

?

Is there a (not necessarily unique) minimal NFA?

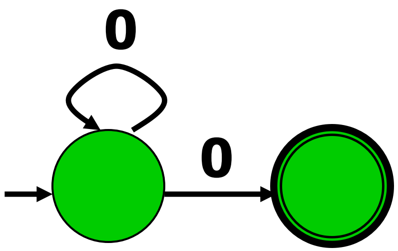
?

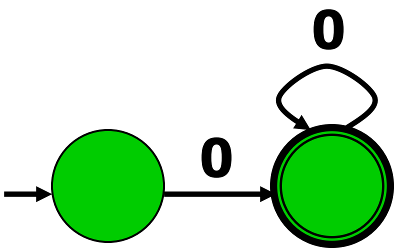
Of course.

NFA Minimization

In general, minimal NFA not unique!

Example NFAs for $\mathbf{0}^+$:





Both minimal, but not isomorphic.

Minimizing DFA’s

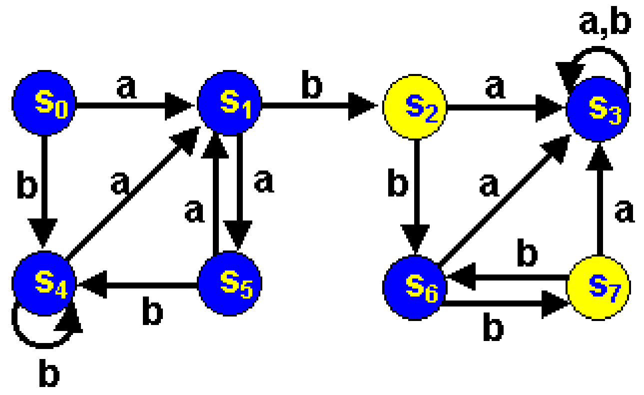
By Partitioning

Minimizing DFA's

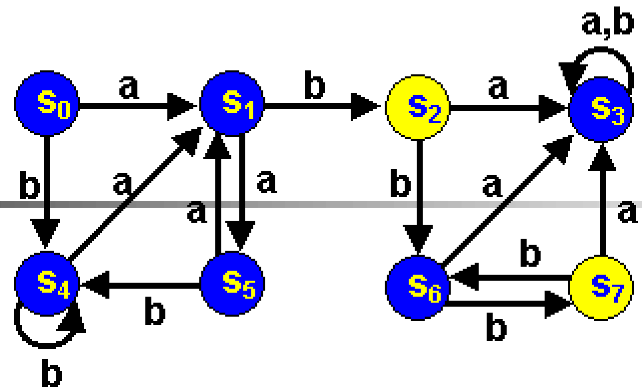
- ***Lots*** of methods
- All involve finding **equivalent states**:
  - States that go to equivalent states under all inputs (sounds recursive)
- We will learn the ***Partitioning Method***

Minimizing DFA's by Partitioning

Consider the following dfa



- Accepting states are yellow
- Non-accepting states are blue
- Are any states really the same?



- S_2 and S_7 are really the same:

  Both Final states

  Both go to S6 under input *b*

  Both go to S3 under an *a*
- S0 and S5 really the same. Why?
- We say each pair is equivalent

Are there any other equivalent states?
We can merge equivalent states into 1 state

Partitioning Algorithm

- First
  - Divide the set of states into

    Final and
    Non-final states

Partition I

Partition II

	a	*b*
S_0	S_1	S_4
S_1	S_5	S_2
S_3	S_3	S_3
S_4	S_1	S_4
S_5	S_1	S_4
S_6	S_3	S_7
$*S_2$	S_3	S_6
$*S_7$	S_3	S_6

Partitioning Algorithm

- Now
  - See if states in each partition each go to the same partition
- S_1 & S_6 are different from the rest of the states in Partition I (but like each other)
- We will move them to their own partition

	a		b	
S_0	S_1	I	S_4	I
S_1	S_5	I	S_2	II
S_3	S_3	I	S_3	I
S_4	S_1	I	S_4	I
S_5	S_1	I	S_4	I
S_6	S_3	I	S_7	II
$*S_2$	S_3	I	S_6	I
$*S_7$	S_3	I	S_6	I

Partitioning Algorithm

	a	*b*
S_0	S_1	S_4
S_5	S_1	S_4
S_3	S_3	S_3
S_4	S_1	S_4
S_1	S_5	S_2
S_6	S_3	S_7
*S_2	S_3	S_6
*S_7	S_3	S_6

Partitioning Algorithm

- Now again
  - See if states in each partition each go to the same partition
  - In Partition I, S_3 goes to a different partition from S_0, S_5 and S_4
  - **We'll move S3 to its own partition**

	a	b
S_0	S_1 III	S_4 I
S_5	S_1 III	S_4 I
S_3	S_3 I	S_3 I
S_4	S_1 III	S_4 I
S_1	S_5 I	S_2 II
S_6	S_3 I	S_7 II
$*S_2$	S_3 I	S_6 III
$*S_7$	S_3 I	S_6 III

Partitioning Algorithm

Note changes in S_6, S_2 and S_7

	a	*b*
S_0	S_1 III	S_4 I
S_5	S_1 III	S_4 I
S_4	S_1 III	S_4 I
S_3	S_3 IV	S_3 IV
S_1	S_5 I	S_2 II
S_6	S_3 IV	S_7 II
*S_2	S_3 IV	S_6 III
*S_7	S_3 IV	S_6 III

Partitioning Algorithm

- **Now S_6 goes to a different partition on an *a* from S_1**
- **S_6 gets its own partition.**
- **We now have 5 partitions**
- **Note changes in S_2 and S_7**

	a	*b*
S_0	S_1 III	S_4 I
S_5	S_1 III	S_4 I
S_4	S_1 III	S_4 I
S_3	S_3 IV	S_3 IV
S_1	S_5 I	S_2 II
S_6	S_3 IV	S_7 II
*S_2	S_3 IV	S_6 V
*S_7	S_3 IV	S_6 V

Partitioning Algorithm

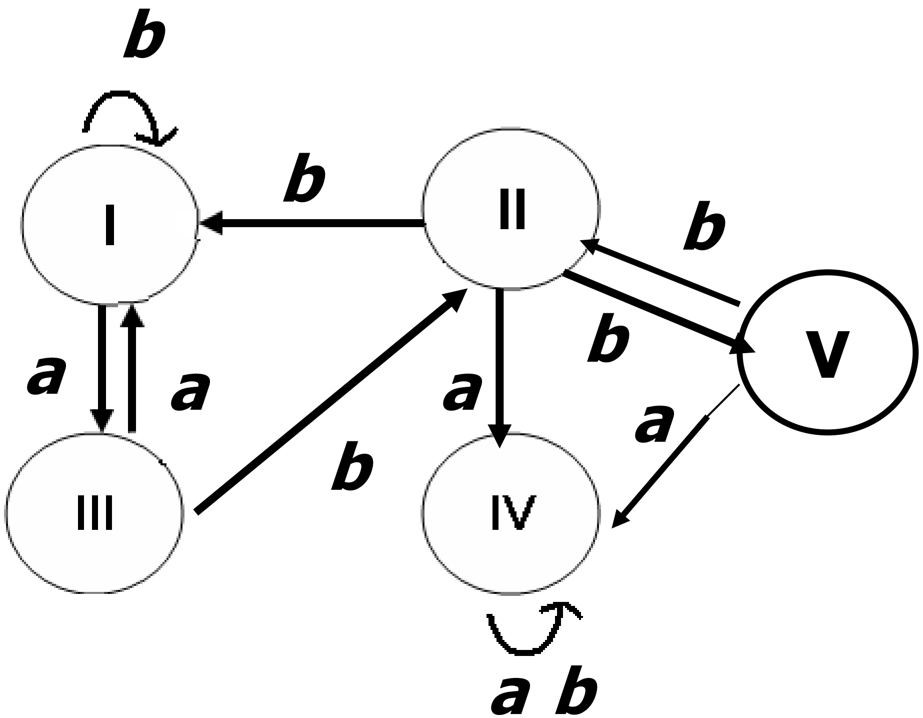
- All states within each of the 5 partitions are identical.
- We might as well call the states I, II III, IV and V.

	a	b
S_0	S_1 III	S_4 I
S_5	S_1 III	S_4 I
S_4	S_1 III	S_4 I
S_3	S_3 IV	S_3 IV
S_1	S_5 I	S_2 II
S_6	S_3 IV	S_7 II
*S_2	S_3 IV	S_6 V
*S_7	S_3 IV	S_6 V

Partitioning Algorithm

Here they are:

	a	*b*
I	III	I
*II	IV	V
III	I	II
IV	IV	IV
V	IV	II


b
I
b
II
b
V
b
a
a
a
a
b
III
IV
a b