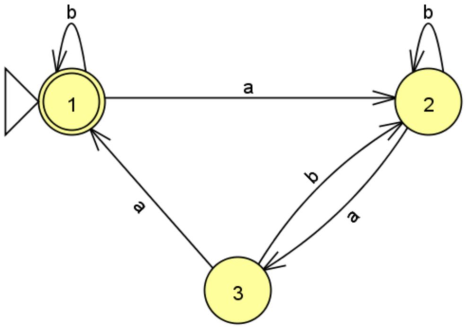
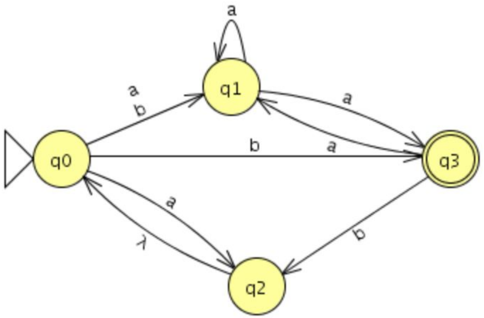
DFA Minimization & Equivalence

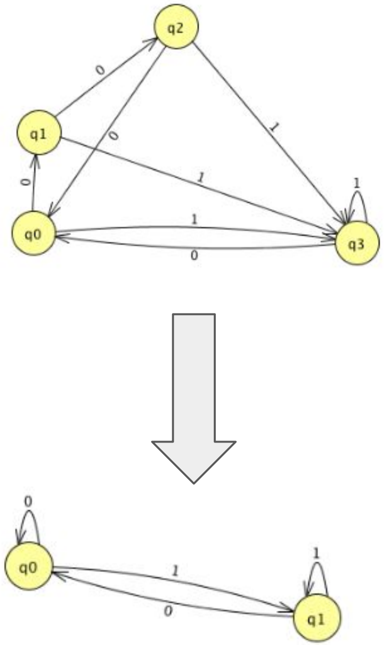
Nathan, Brandon, and David

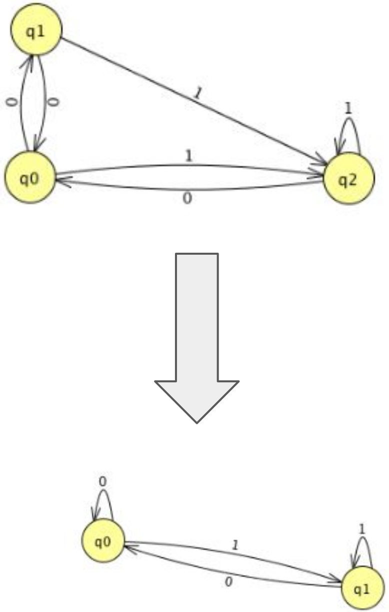


!=



DFA Minimization





L(N) = { the set of strings w such that w ends with a 1 }

Steps Involved

- Delete all nodes in the graph that are unreachable
- Run the table filling algorithm (TFS) on the graph
- Collect coordinates for unmarked cells
- Use Union Find to find all the pairs and group them into disjoint groups (grouping together all the remaining pairs into a new group)
- Use the disjoint sets to make a single state pairs {1, 3, 5} => {1_3_5}
- Use the single state pairs to access the original graph using a slightly modified version of NFA to DFA conversion to create the newly minimized dfa

Steps Visualized

State Graph
[1, 3]
[2, 3]
[0, 3]
[0, 3]

TFS
[3, 0, 0, 0]
[0, 3, 0, 0]
[0, 0, 3, 0]
[1, 1, 1, 3]

Empty Cells
[(0, 1),
(0, 2),
(1, 2)]

Same States
[[0, 1, 2]]

Min DFA States
['0_1_2']

Equivalent States
{[0, 1, 2]}

Nonequivalent States
{[3]}

Combined Min States
['0_1_2', '3']

Steps Visualized (cont.)

Original
DFA / NFA
[[1], [3]]
[[2], [3]]
[[0], [3]]
[[0], [3]]

Final DFA
0 [0, 1]
1 [0, 1]

Minimized DFA
0_1_2 ['0_1_2', '3']
3 ['0_1_2', '3']

DFA Equivalence

Problem Statement:

"Design an efficient algorithm to test the equivalence of DFA's"

Data Structure/Algorithms Used:

Hopcroft-Karp Algorithm using a Union/Find Data Structure

Union/Find Data Structure

- Sometimes called a Disjoint Set Data Structure
- Two main operations:
  - FIND(x): returns the set identifier that x belongs to
  - UNION(x,y): combines set x and y into a single set
- Our Implementation:
  - A simple array based storage where index = the identifier for the particular element, and the value at the index = the identifier for the set that the element belongs to.
  - Takes advantage of “path compression” to make sure the find operation remains constant time.

Hopcroft-Karp Algorithm

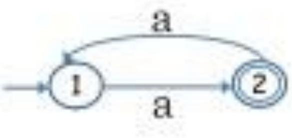
Main Steps:

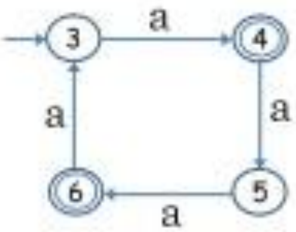
1. Make a set for every state in the union of the two DFA’s
2. Union the starting states into one set and push onto a stack
3. Pop from stack if not empty and search for the sets which contain the states reached by the transitions from the popped pair; if the sets equal each other do nothing, otherwise union the two different sets into one set
4. The DFAs are considered equivalent if and only if no set contains both a final and a non-final state

Visual Representation

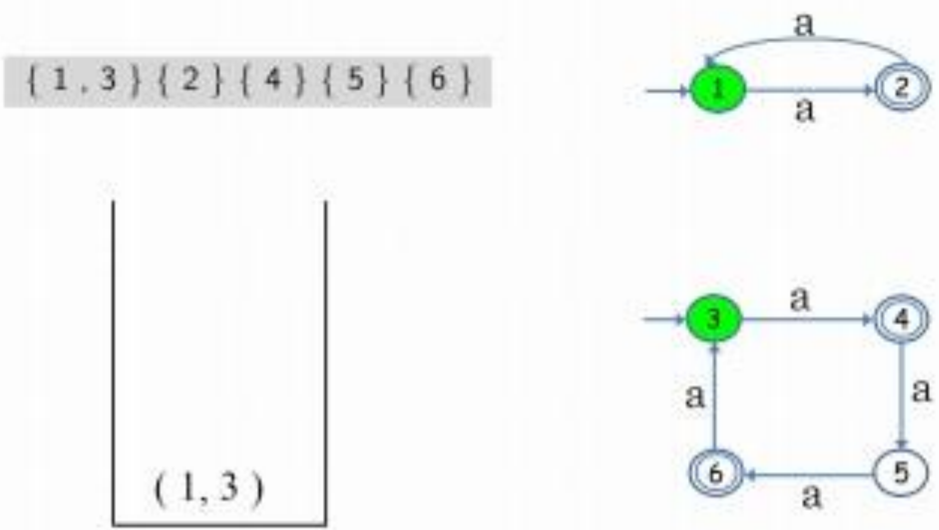
Example provided by: Bhanva Ganji and Gurpreet Kaur: “A Linear Algorithm For Testing Equivalence of Finite Automata”
http://drona.csa.iisc.ernet.in/~deepakd/atc-2010/equivalence-checking-seminar.pdf

{ 1 } { 2 } { 3 } { 4 } { 5 } { 6 }

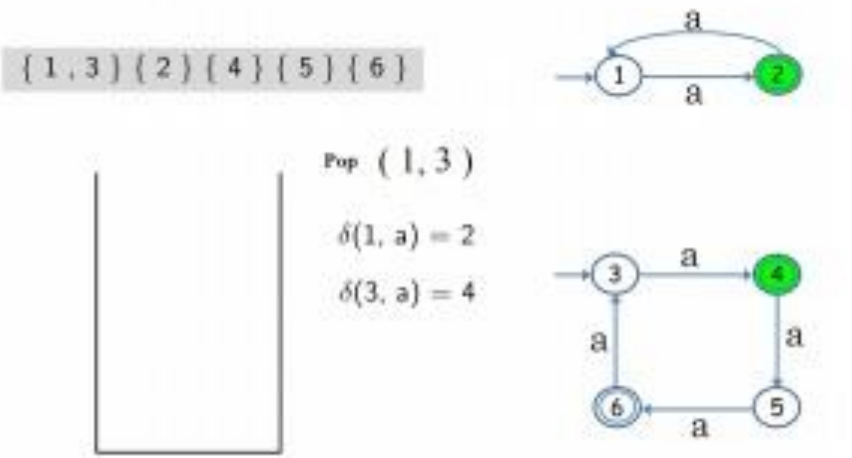




Step 1 : For every state $q \in Q_1 \cup Q_2$, MakeSet(q)



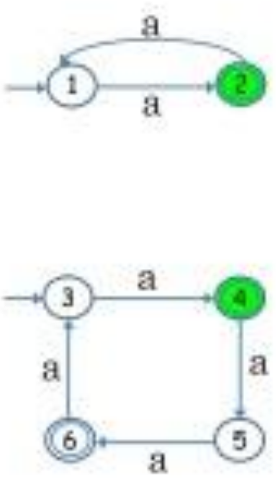
Step 2 : Union(s_1 , s_2) and Push (s_1 , s_2) on to a stack, S



Step 3 : While S is non-empty

- Pop pair (q_1 , q_2) from S
- For each a $\in \Sigma$
  - r_1 = Find($\delta(q_1$, a)) and r_2 = Find($\delta(q_2$, a))
  - If $r_1 \neq r_2$
    Union(r_1 , r_2) and Push (r_1 , r_2) on to S

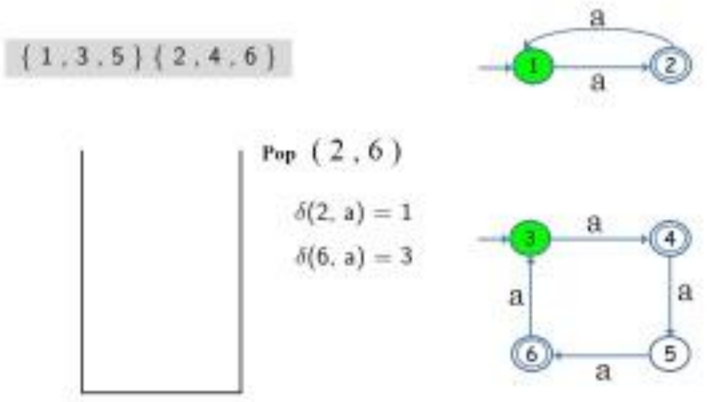
{ 1 , 3 } { 2 , 4 } { 5 } { 6 }



Step 3 : While S is non-empty

- Pop pair (q_1 , q_2) from S
- For each a $\in \Sigma$
  - r_1 = Find($\delta(q_1, \text{a})$) and r_2 = Find($\delta(q_2, \text{a})$)
  - If $r_1 \neq r_2$
    Union(r_1 , r_2) and Push (r_1 , r_2) on to S

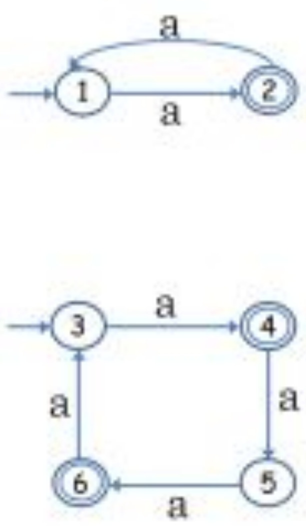
Skipping to final step:



Step 3 : While S is non-empty

- Pop pair (q_1 , q_2) from S
- For each a $\in \Sigma$
  - $r_1 = \text{Find}(\delta(q_1, \text{a}))$ and $r_2 = \text{Find}(\delta(q_2, \text{a}))$
  - If $r_1 \neq r_2$
    Union(r_1 , r_2) and Push (r_1 , r_2) on to S

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



Step 4 : $L(M_1) = L(M_2)$ if and only if no set contains both a final and a non-final state.

Time Complexity

Step 1: O(n)

Step 2:O(1), achieved through the UNION/FIND Data Structure

Step 3: O(mn)

Step 4: O(n)

‘n’ is the number of states of the two DFAs and ‘m’ is the number of symbols in sigma.