Implementation and visualization of the procedure NFA into DFA

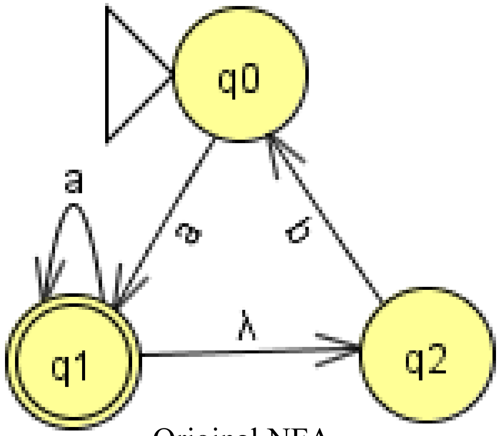
John Cameron

Jordano Baer

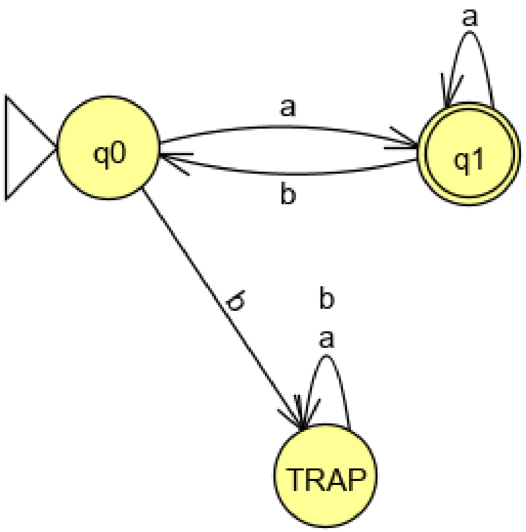
Differences NFA and DFA

1) NFA permits λ-transitions
2) NFA is able to have undeclared or undefined transitions
3) NFA may contain transitions to different states after receiving a single input (can be in multiple states at once)

Two automatas are equivalent if and only if they both accept the same language



Original NFA



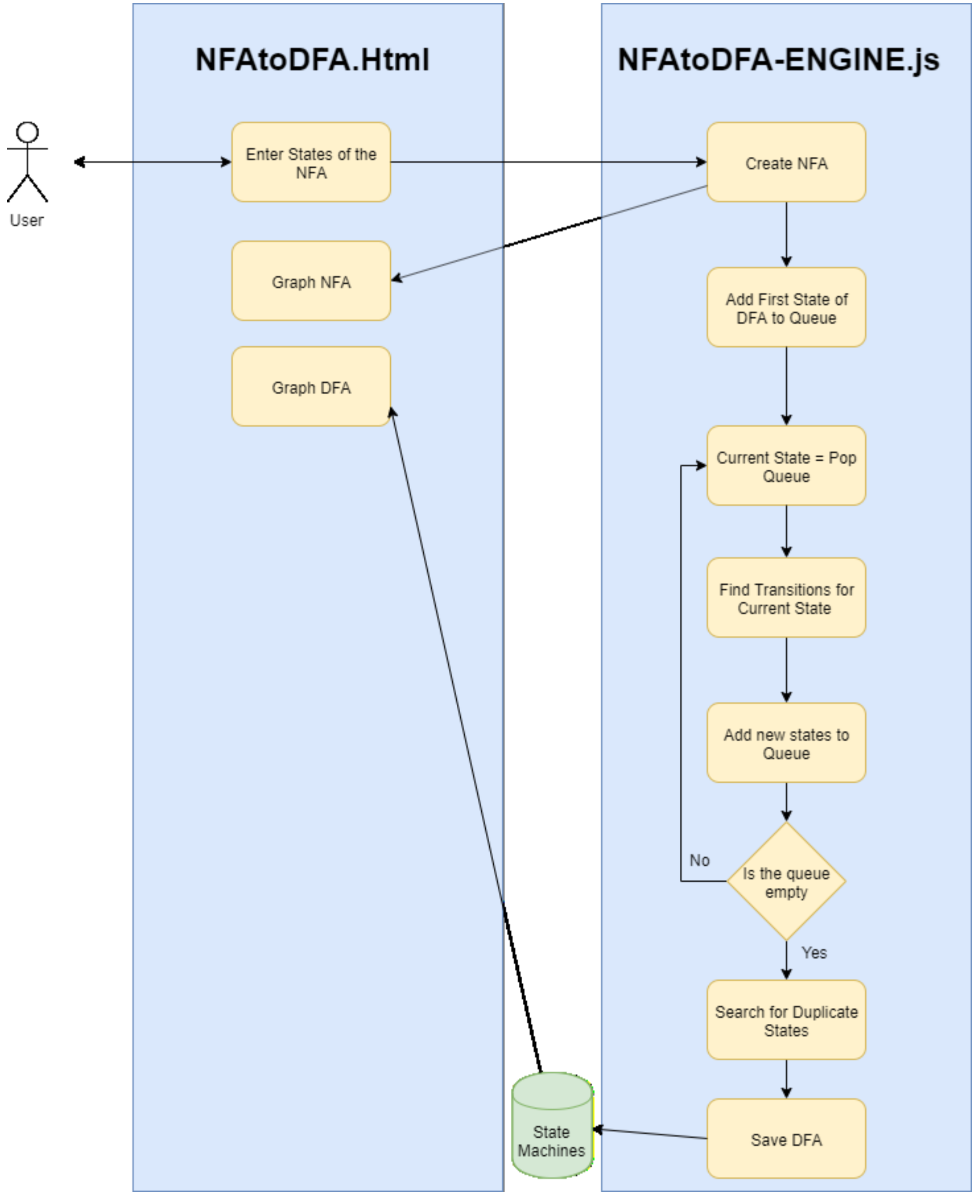
Equivalent DFA

NFA-to-DFA

- Important factors that the algorithm should be aware about during the conversion process:

1) Is a trap state required?
2) What is the initial state?
3) What are the final states?
4) Does the original NFA contain λ-transitions?
5) How will the algorithm know when it is done?
6) Is there a λ-transition from initial state to final state?

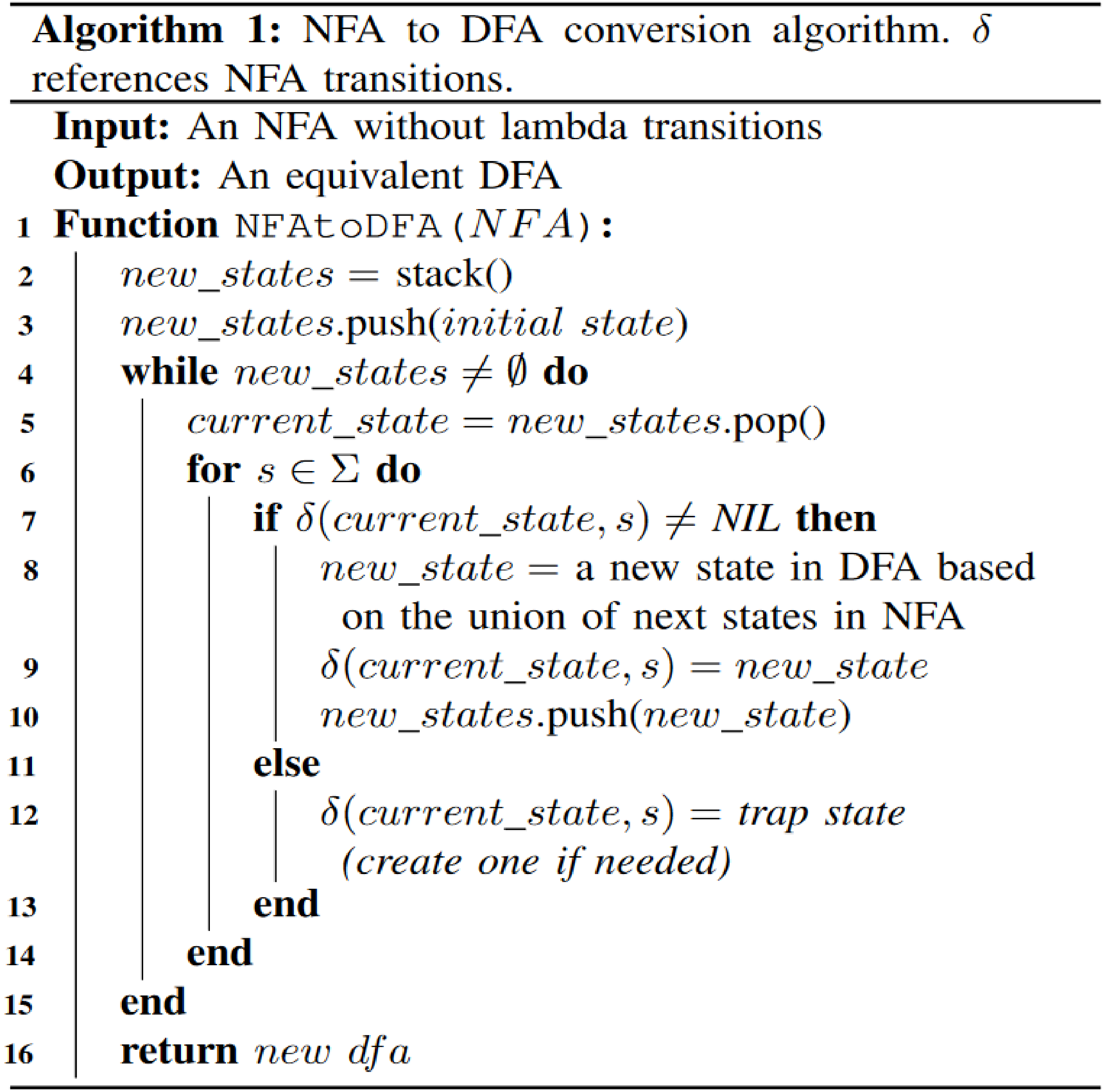
Software Architecture



Pseudocode

1. User enter the original NFA
2. The initial state of the DFA is the closure of the initial state of the original NFA
3. Add the new states of the DFA to a queue
4. While the queue is not empty, find the transitions to the new states (If there’s a new state, add it to the queue)
5. Pop the next state from the queue
6. After the queue is empty, create the graph of the DFA

The new transitions for the DFA are created by the union operation with the closure of the transitions from the original NFA for the current state in the DFA.



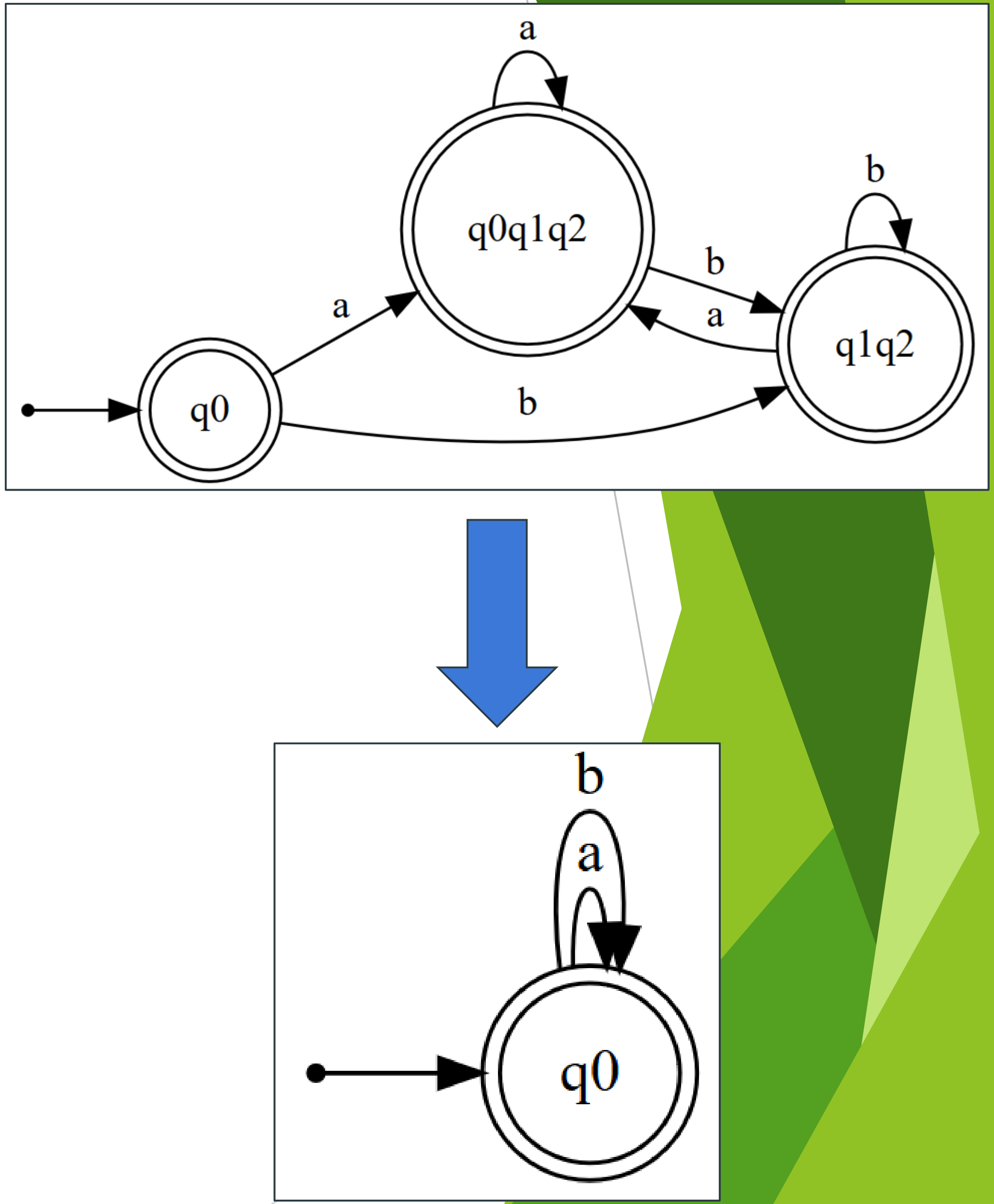
Algorithm 1: NFA to DFA conversion algorithm. δ references NFA transitions.

```
Input: An NFA without lambda transitions
Output: An equivalent DFA
Function NFAtoDFA($NFA$):
    $new\_states$ = stack()
    $new\_states$.push($initial\ state$)
    while $new\_states \neq \emptyset$ do
        $current\_state = new\_states$.pop()
        for $s \in \Sigma$ do
            if $\delta(current\_state, s) \neq NIL$ then
                $new\_state$ = a new state in DFA based
                  on the union of next states in NFA
                $\delta(current\_state, s) = new\_state$
                $new\_states$.push($new\_state$)
            else
                $\delta(current\_state, s) = trap\ state$
                  $(create\ one\ if\ needed)$
            end
        end
    end
    return $new\ dfa$
```

Conversion algorithm is based off the NFA and DFA equivalence theorem

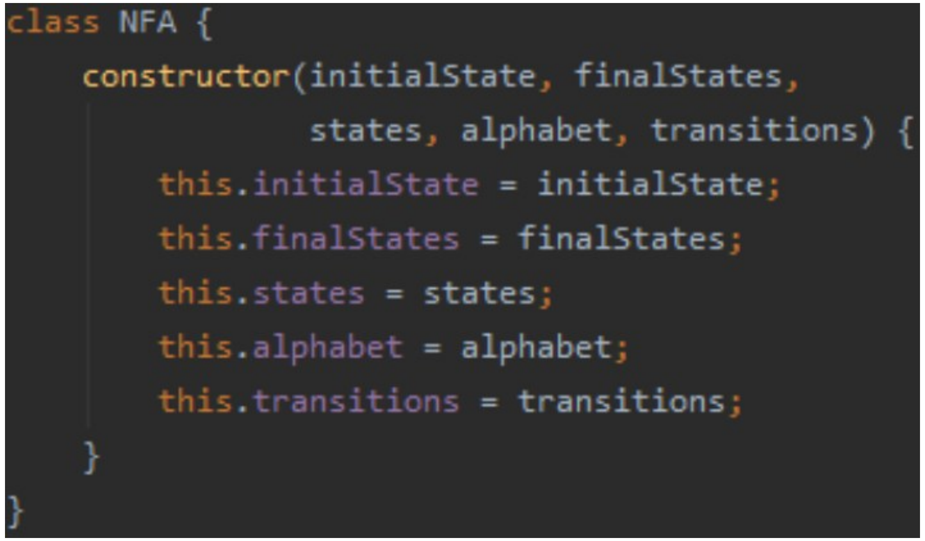
Minimization

- Search for states that have the same transitions for every input in the alphabet
- If both states are final or not final, replace state1 with state2 in the DFA
- Delete state2
- If the initial state is marked to be removed, select the other state instead



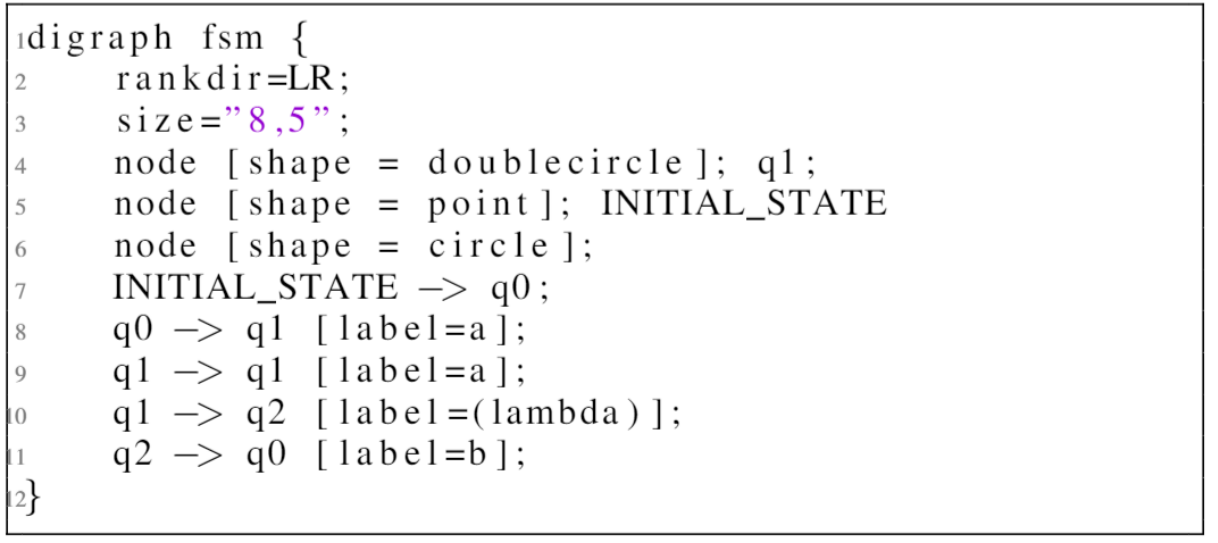
Visualization

- Convert our Standard NFA into the digraph format accepted by the Graphviz library



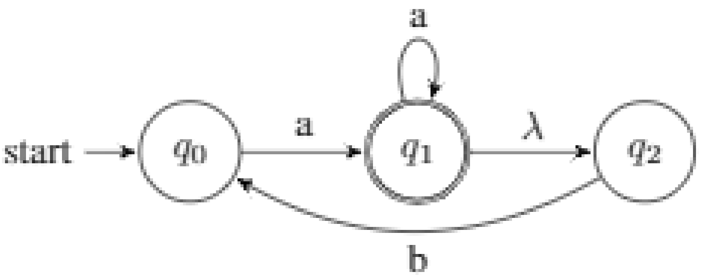
```
class NFA {
    constructor(initialState, finalStates,
                states, alphabet, transitions) {
        this.initialState = initialState;
        this.finalStates = finalStates;
        this.states = states;
        this.alphabet = alphabet;
        this.transitions = transitions;
    }
}
```

Custom NFA

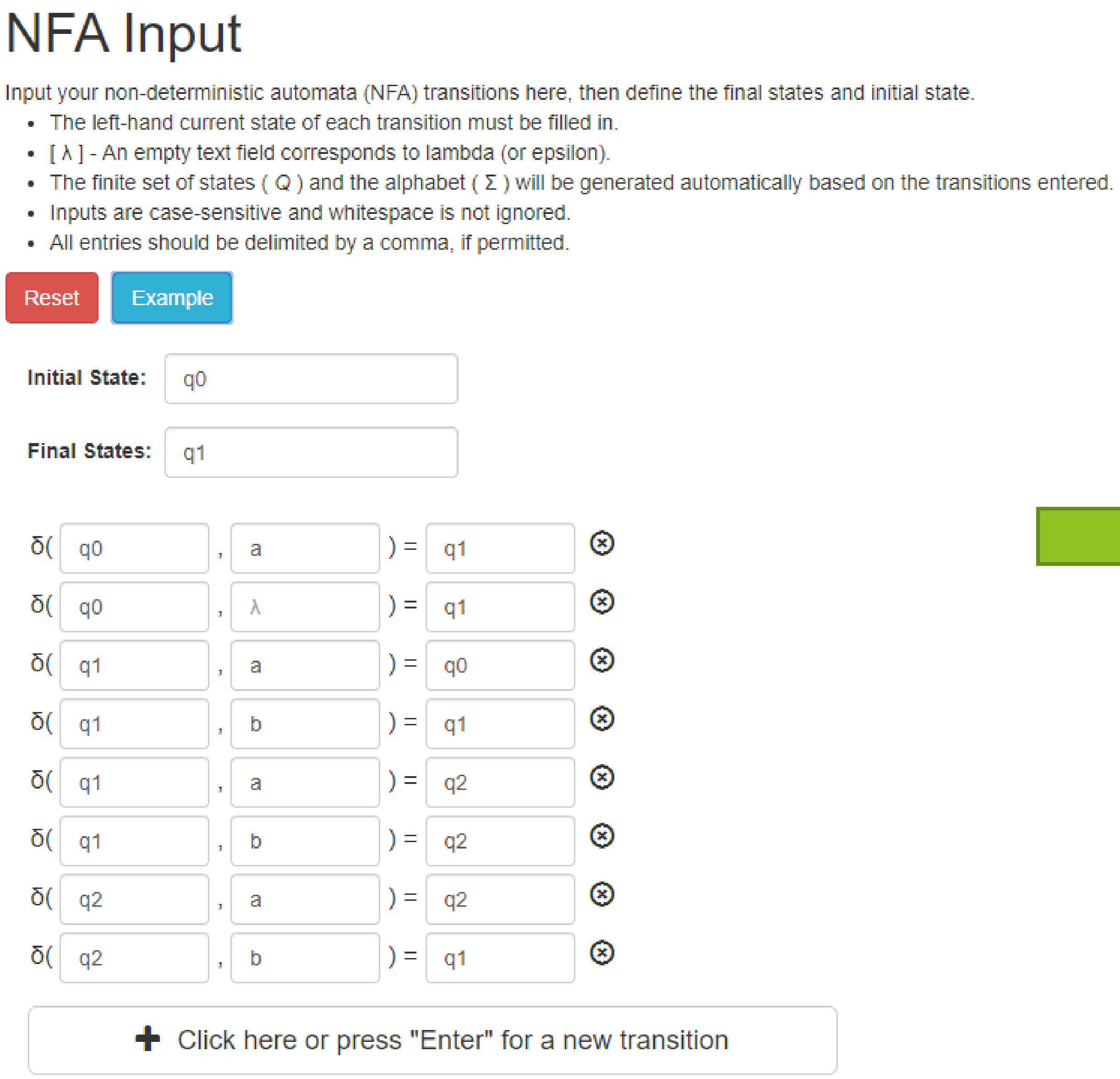


```
digraph fsm {
    rankdir=LR;
    size=”8,5”;
    node [shape = doublecircle]; q1;
    node [shape = point]; INITIAL_STATE
    node [shape = circle];
    INITIAL_STATE –> q0;
    q0 –> q1 [label=a];
    q1 –> q1 [label=a];
    q1 –> q2 [label=(lambda)];
    q2 –> q0 [label=b];
}
```

VIZ library digraph format
(Saved in text file)



Example



NFA Input

Input your non-deterministic automata (NFA) transitions here, then define the final states and initial state.

- The left-hand current state of each transition must be filled in.
- [λ] - An empty text field corresponds to lambda (or epsilon).
- The finite set of states (Q) and the alphabet (Σ) will be generated automatically based on the transitions entered.
- Inputs are case-sensitive and whitespace is not ignored.
- All entries should be delimited by a comma, if permitted.

Initial State: q0

Final States: q1

δ(q0 , a) = q1

δ(q0 , λ) = q1

δ(q1 , a) = q0

δ(q1 , b) = q1

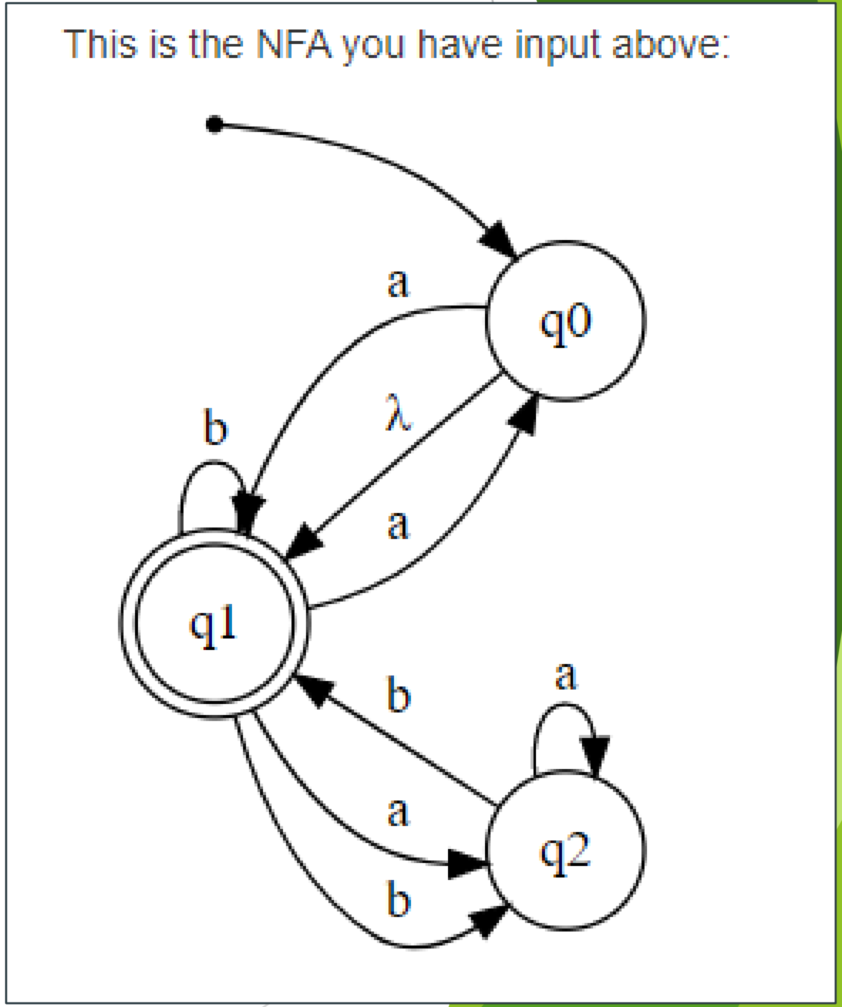
δ(q1 , a) = q2

δ(q1 , b) = q2

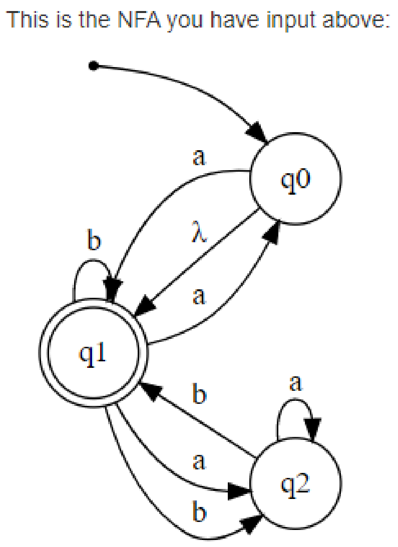
δ(q2 , a) = q2

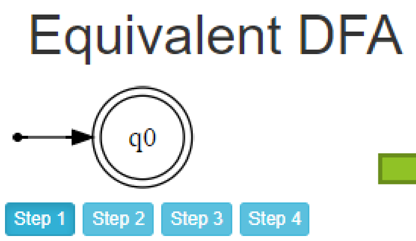
δ(q2 , b) = q1

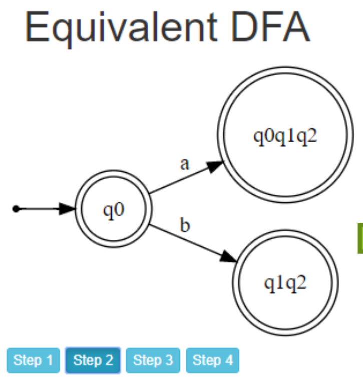
This is the NFA you have input above:

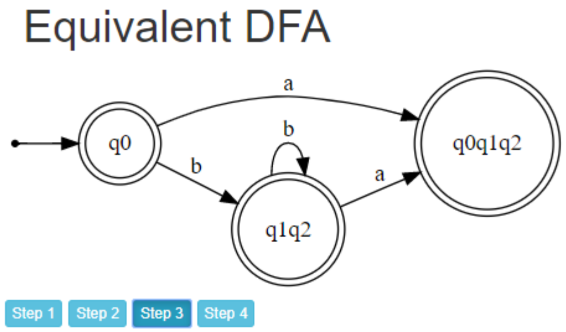


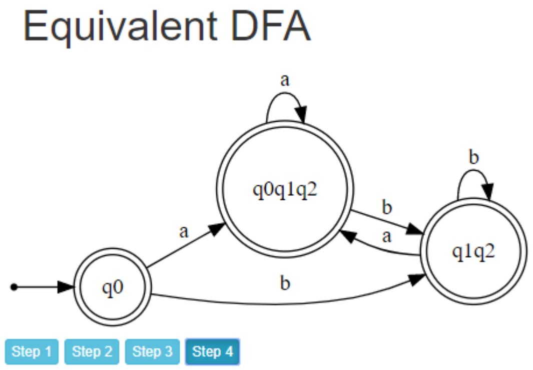
NFA-to-DFA Procedure


This is the NFA you have input above:

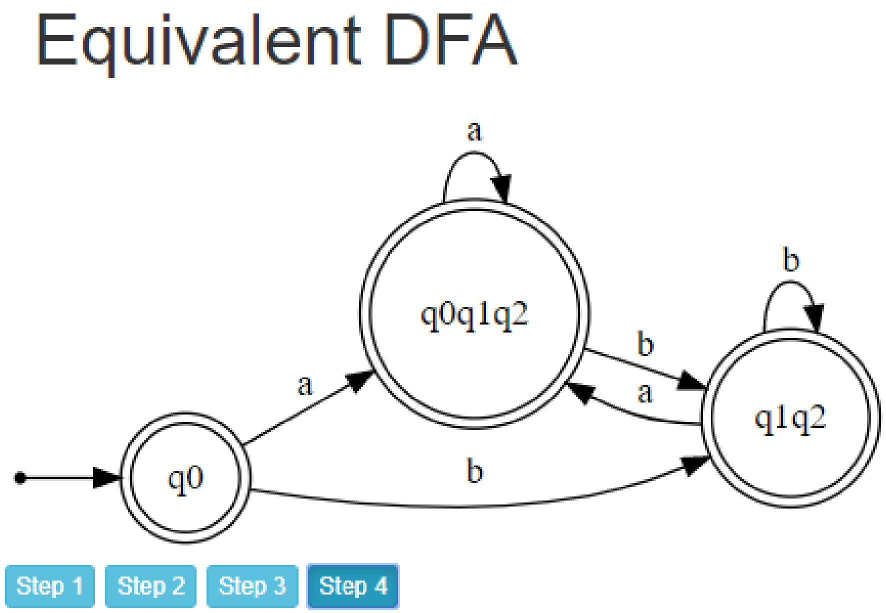

Equivalent DFA


Equivalent DFA


Equivalent DFA

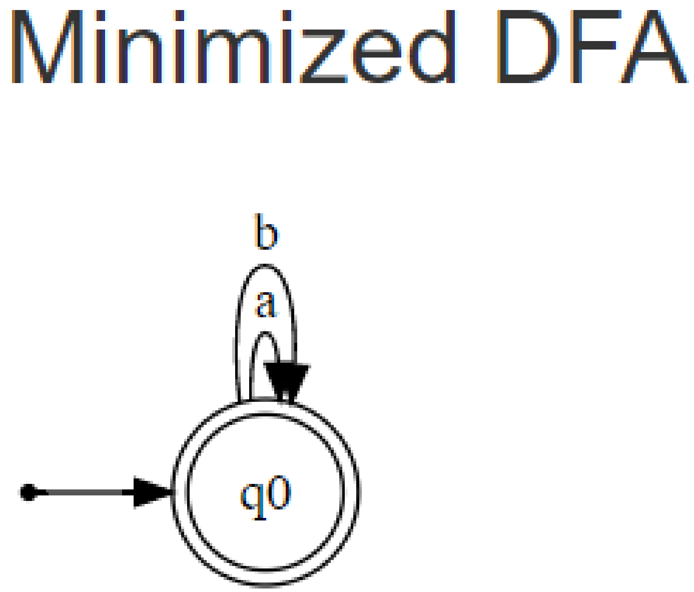

Equivalent DFA

Reduction



Equivalent DFA

Complete DFA



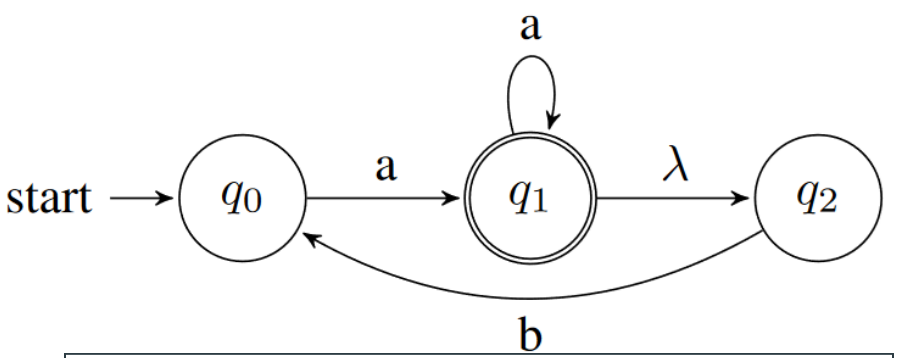
Minimized DFA

Reduced DFA

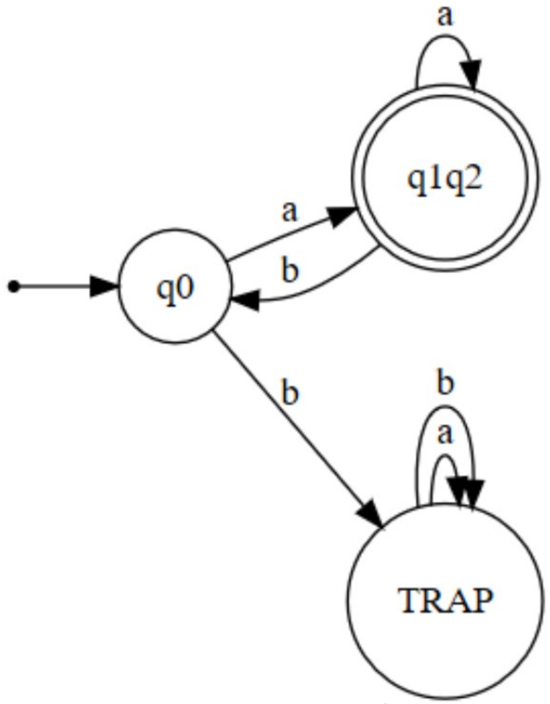
Testing for Implementation Correctness

- Multiple NFAs were used to test the correctness of our implementation

- We know that the conversion process is successful if the DFA accepts the same language as the NFA



$$\begin{tikzcd}
q0 \arrow[r, "a"] & q1 \arrow[r, "\lambda"] \arrow[dr, "b"'] & q2 \arrow[r, "b"] & q0 \arrow[r, "a"] & q1 \\
 & & \emptyset & &
\end{tikzcd}$$

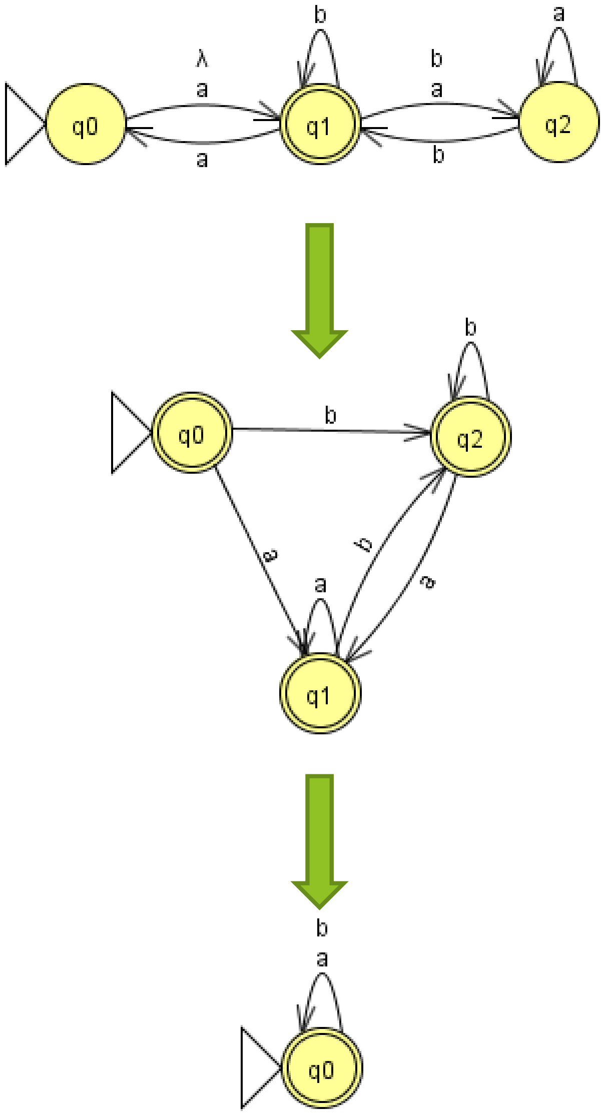


$$q0 \xrightarrow{a} q1 \xrightarrow{b} q0 \xrightarrow{a} q1q2$$

References

- Linz, Peter. An Introduction to Formal Languages and Automata. Jones & Bartlett Learning, 2012.
- “Vis.js Community Edition *.” Visjs.org, https://visjs.org/
- “JFLAP.” JFLAP, http://www.jflap.org/

Demonstration



Worksheet #4

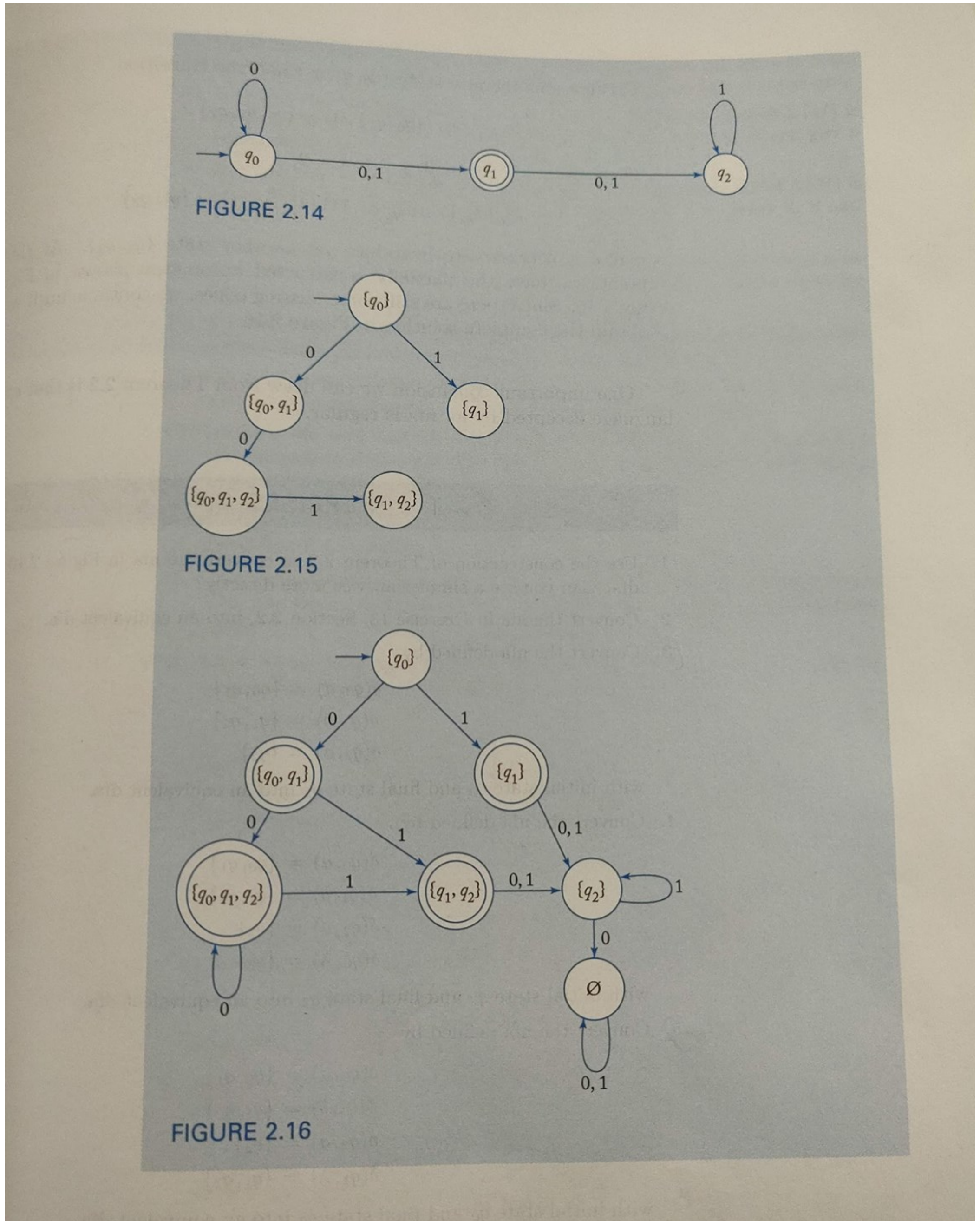
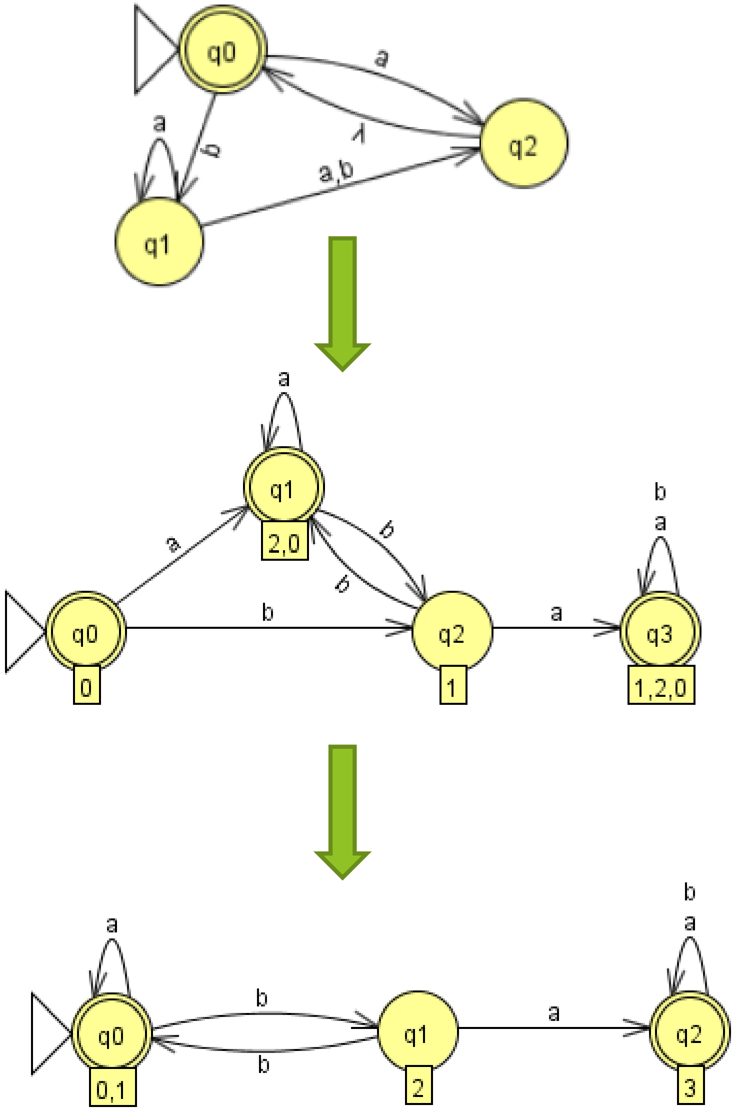


FIGURE 2.14

FIGURE 2.15

FIGURE 2.16

Textbook Page 63



Homework 2