A Linear Algorithm for Testing Equivalence of Finite Automata

Namrata Jain

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

October 30, 2013

Outline

Plan

A Quick Recap

DFA over Σ : $M = (Q, s, \delta, F)$

Q *is a finite set of states*

$s \in Q$ *represents the start state*

$\delta : Q \times \Sigma \to Q$ *is the transition function*

$F \subseteq Q$ *is the set of final states*

Define $\hat{\delta} : Q \times \Sigma^* \to Q$

- $\hat{\delta}(q, \epsilon) = q$
- $\hat{\delta}(q, w \cdot a) = \delta(\hat{\delta}(q, w), a)$

Language accepted by DFA M (Denoted by $L(M)$*)*

$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(s, w) \in F\}$$

Problem Definition

Input : 2 DFA's over Σ

- $M_1 = (Q_1, s_1, \delta_1, F_1)$
- $M_2 = (Q_2, s_2, \delta_2, F_2)$

Output : Is $L(M_1) = L(M_2)$?

$$\forall w \in \Sigma^*, \quad \hat{\delta_1}(s_1, w) \in F_1 \ \ \textit{iff} \ \ \hat{\delta_2}(s_2, w) \in F_2$$

Existing Solutions

- Previous algorithms have a time complexity of
  1. $O(n^2)$
  2. $O(n\ lgn)$
- Hopcroft-Karp algorithm has a time complexity of $O(n|\Sigma|)$

$$n = |Q_1| + |Q_2|$$

Plan

Notation

Equivalent States

Two states p and q are said to be equivalent ($p \equiv q$) if

$$\forall p, q \in Q_1 \bigcup Q_2 \;\; \forall w \in \Sigma^*,$$

$$\hat{\delta}(p, w) \in F_1 \bigcup F_2 \; iff \; \hat{\delta}(q, w) \in F_1 \bigcup F_2$$

Right invariant Equivalence Relation

A equivalence relation $\equiv$ over $Q_1 \bigcup Q_2$ is right invariant if

$$\forall p, q \in Q_1 \bigcup Q_2 \;\; \forall a \in \Sigma,$$

$$\delta(p, a) \equiv \delta(q, a)$$

Intuition

- $L(M_1) = L(M_2)$
  $\Longrightarrow$ *s_1 and s_2 are equivalent*
  $\Longrightarrow \delta(s_1, a) = \delta(s_2, a)$
- We begin by assuming s_1 and s_2 equivalent.
- Sets are merged whenever it is found two states need to be equivalent for the assumption to hold.
- When the process terminates, M_1 and M_2 are equivalent if none of the sets has a final and a non-final state simultaneously.

Data Structure Used

- Data Structure used is a linear list of sets of elements. Each list has a name.
- It can execute only two types of instructions
  1. **FIND(x)** : It returns the name of the set containing x
  2. **MERGE(A, B, C)** : It merges set A and B and names it C
- A sequence of n instructions takes $O(n)$ time.

Algorithm

1. Initialize Data Structures
   a. $\forall q \in Q_1 \bigcup Q_2$, create and initalize a set in Linear List with name q
   b. $Stack = \phi$
2. Assume s_1 and s_2 to be equivalent
   a. MERGE(s_1, s_2, s_2)
   b. Push(s_1, s_2)
3. Repet until stack is empty
   a. Pop (q_1, q_2)
   b. $r_1 = FIND(\delta(q_1, a))$
   c. $r_2 = FIND(\delta(q_2, a))$
   d. if $r_1 \neq r_2$
      i. MERGE(r_1, r_2, r_2)
      ii. Push(r_1, r_2)
4. Check if equivalent

   Scan states on each list. Output "TRUE" iff no list contains a final and a non-final state and "FALSE" otherwise

Example 1 : Step 1

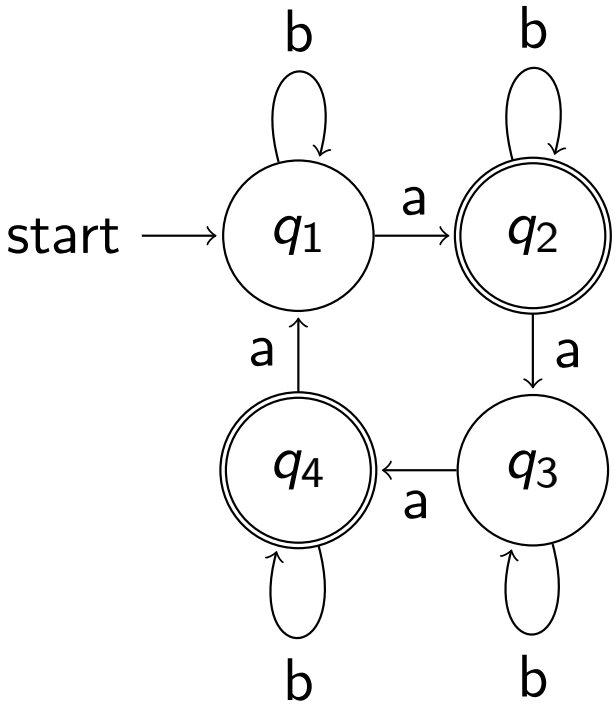

Figure 1 : DFA 1

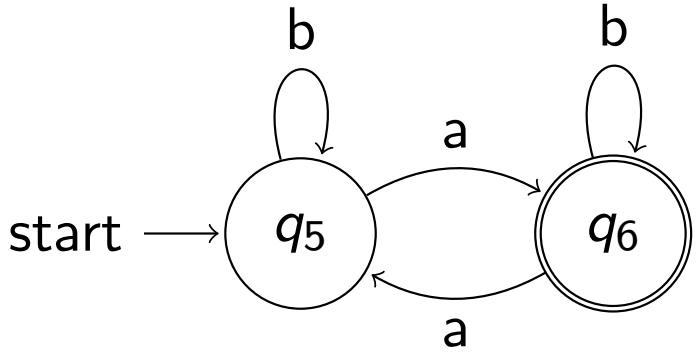

Figure 2 : DFA 2

Stack : ϕ

Linear List : $\{q_1\}, \{q_2\}, \{q_3\}, \{q_4\}, \{q_5\}, \{q_6\}$

Figure 3 : Stack and Linear List

Example 1 : Step 2

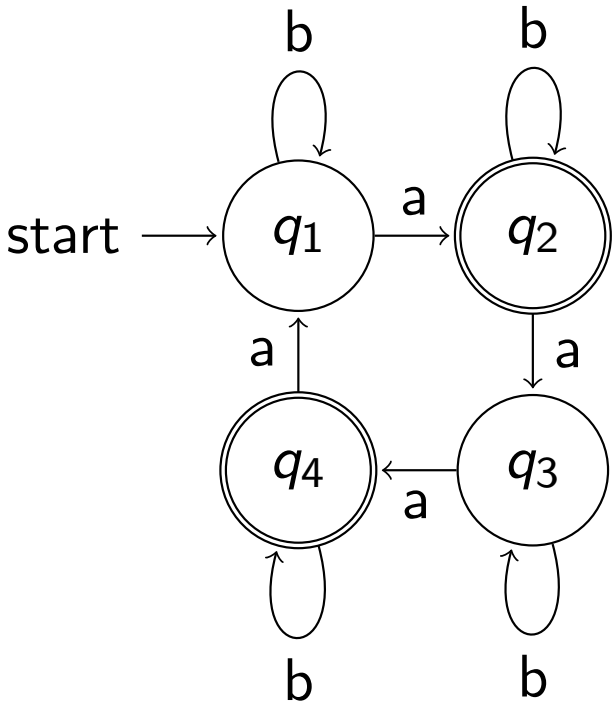

Figure 1 : DFA 1

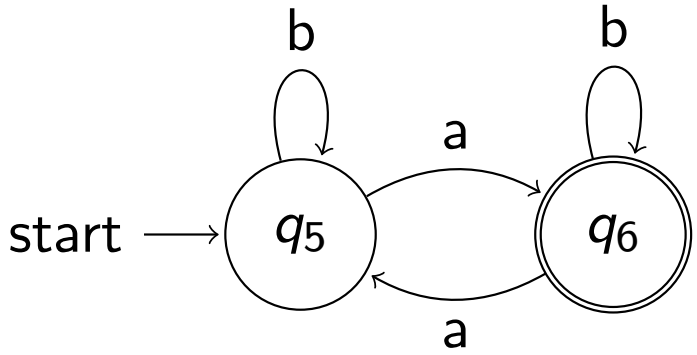

Figure 2 : DFA 2

Stack : $\{q_1, q_5\}$

Linear List : $\{q_1, q_5\}$, $\{q_2\}$, $\{q_3\}$, $\{q_4\}$, $\{q_6\}$

Figure 3 : Stack and Linear List

Example 1 : Step 3

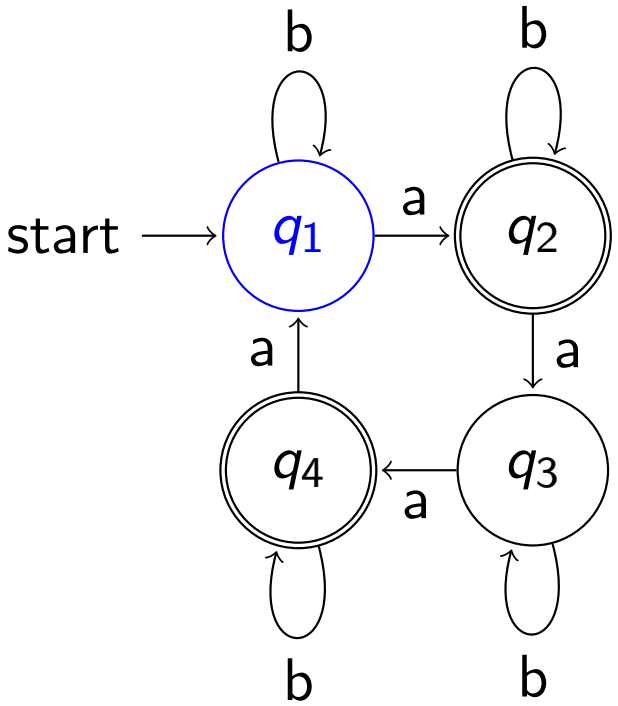

Figure 1 : DFA 1

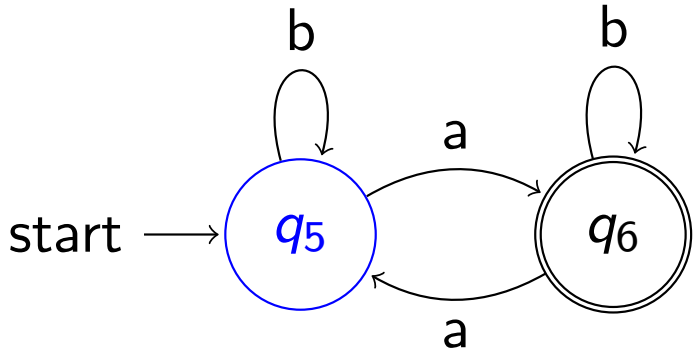

Figure 2 : DFA 2

Stack : $\{q_2, q_6\}$

Linear List : $\{q_1, q_5\}$, $\{q_2, q_6\}$, $\{q_3\}$, $\{q_4\}$

Figure 3 : Stack and Linear List

Example 1 : Step 3

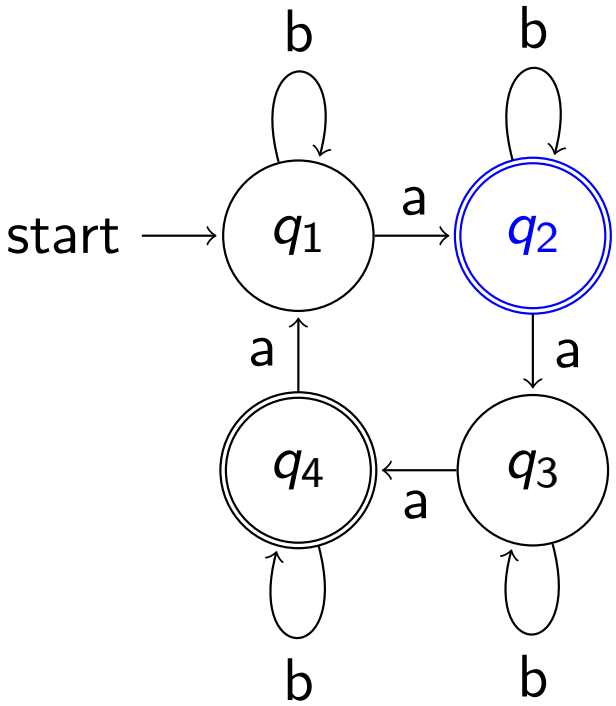

Figure 1 : DFA 1

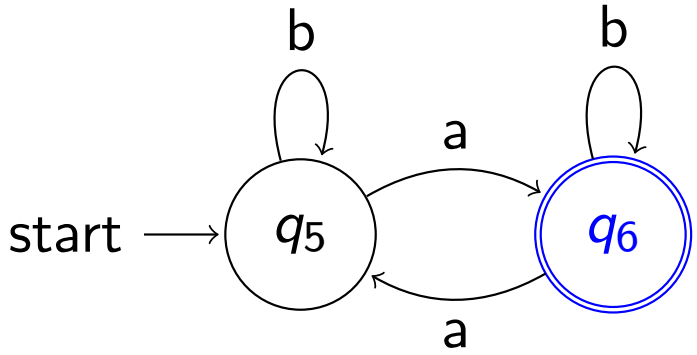

Figure 2 : DFA 2

Stack : $\{q_3, q_5\}$

Linear List : $\{q_1, q_3, q_5\}$, $\{q_2, q_6\}$, $\{q_4\}$

Figure 3 : Stack and Linear List

Example 1 : Step 3

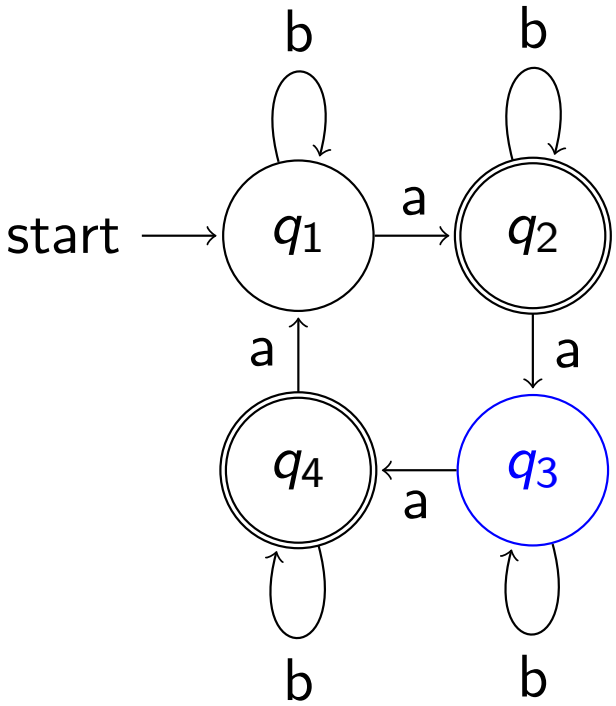

Figure 1 : DFA 1

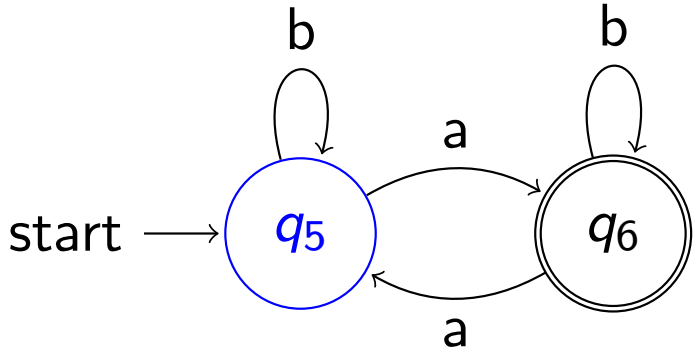

Figure 2 : DFA 2

Stack : $\{q_4, q_6\}$

Linear List : $\{q_1, q_3, q_5\}$, $\{q_2, q_4, q_6\}$

Figure 3 : Stack and Linear List

Example 1 : Step 3

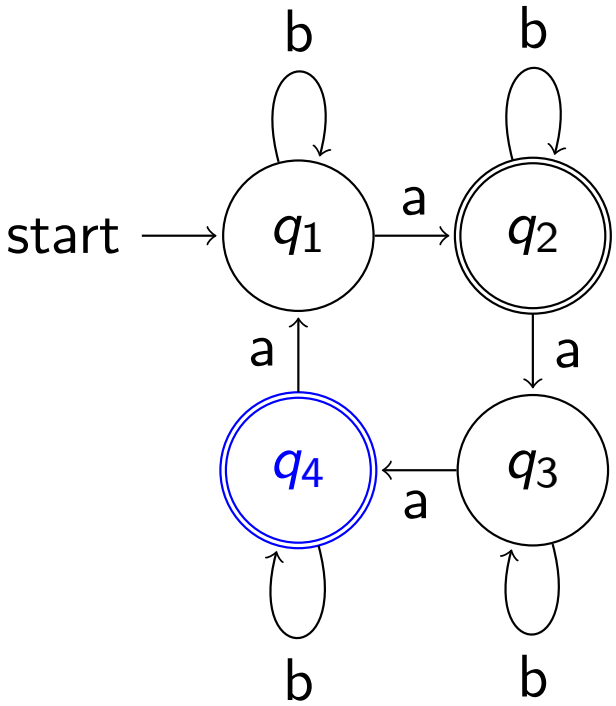

Figure 1 : DFA 1

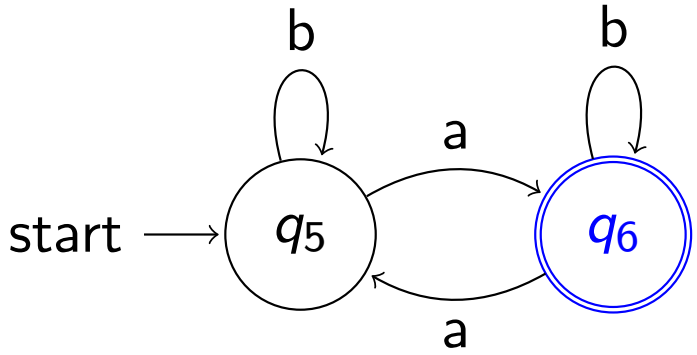

Figure 2 : DFA 2

Stack : ϕ

Linear List : $\{q_1, q_3, q_5\}, \{q_2, q_4, q_6\}$

Figure 3 : Stack and Linear List

Example 2 : Step 1

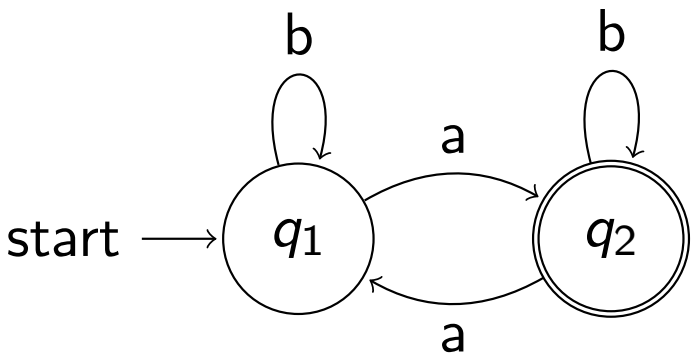

Figure 1 : DFA 1

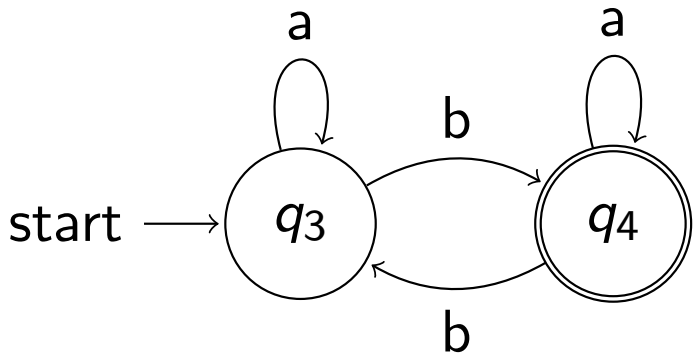

Figure 2 : DFA 2

Stack : ϕ

Linear List : $\{q_1\}, \{q_2\}, \{q_3\}, \{q_4\}$

Figure 3 : Stack and Linear List

Example 2 : Step 2

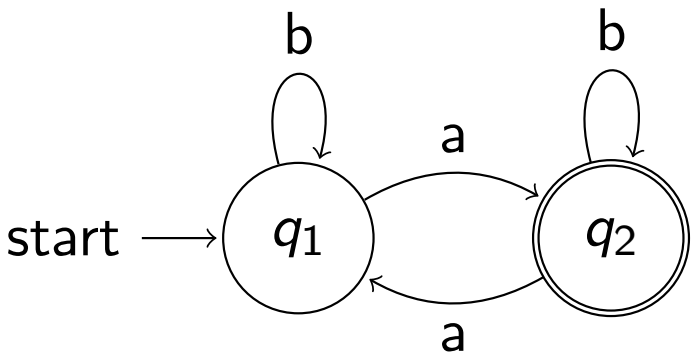

Figure 1 : DFA 1

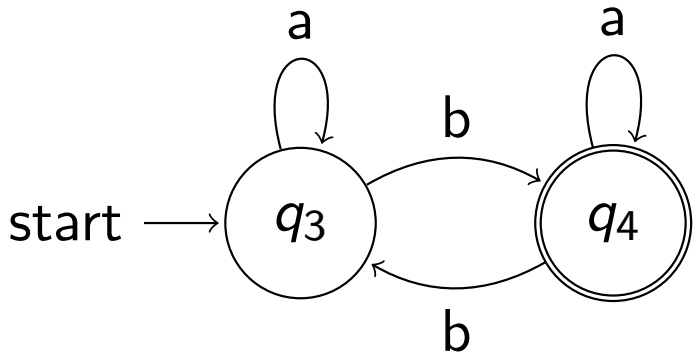

Figure 2 : DFA 2

Stack : $\{q_1, q_3\}$
Linear List : $\{q_1, q_3\}$, $\{q_2\}$, $\{q_4\}$

Figure 3 : Stack and Linear List

Example 2 : Step 3

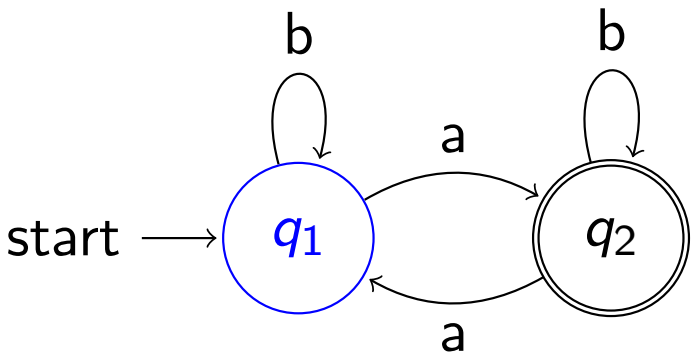

Figure 1 : DFA 1

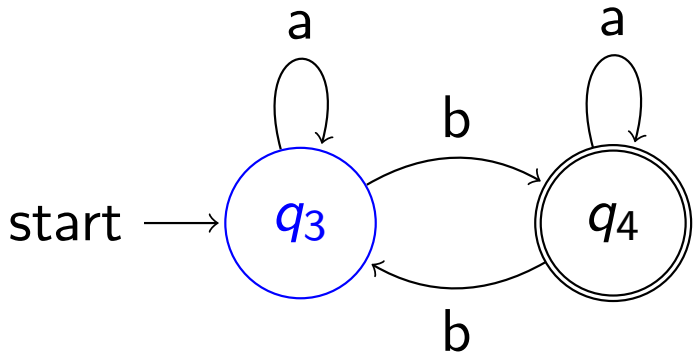

Figure 2 : DFA 2

Stack : $\{q_2, q_3\}, \{q_1, q_4\}$
Linear List : $\{q_1, q_2, q_3, q_4\}$

Figure 3 : Stack and Linear List

Example 2 : Step 3

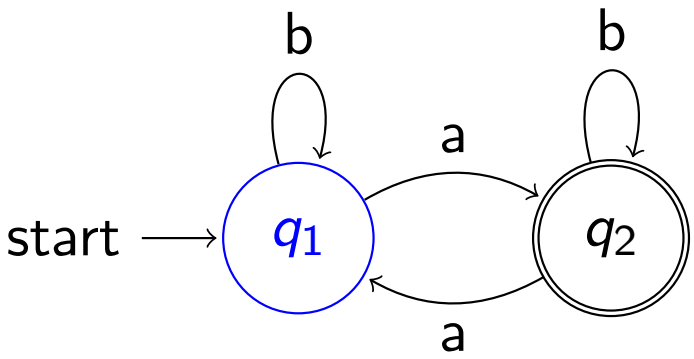

Figure 1 : DFA 1

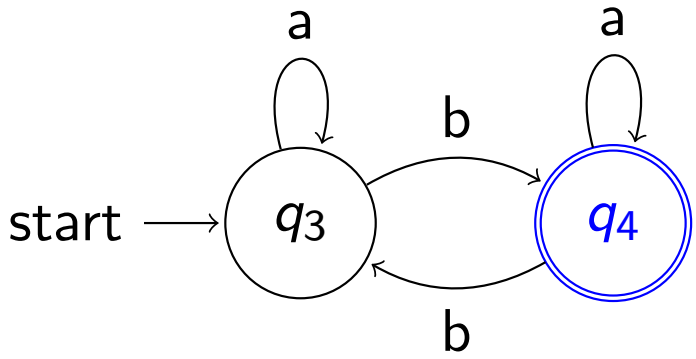

Figure 2 : DFA 2

Stack : $\{q_2, q_3\}$
Linear List : $\{q_1, q_2, q_3, q_4\}$

Figure 3 : Stack and Linear List

Example 2 : Step 3

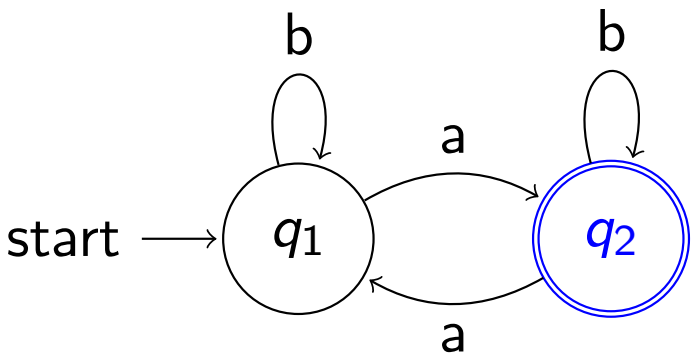

Figure 1 : DFA 1

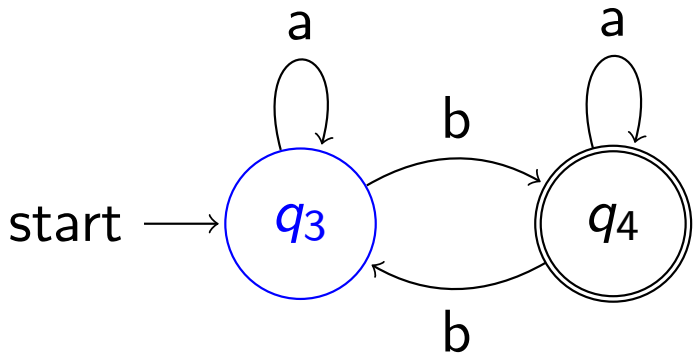

Figure 2 : DFA 2

Stack : ϕ

Linear List : $\{q_1, q_2, q_3, q_4\}$

Figure 3 : Stack and Linear List

Plan

Notation

Connecting Sequence

A sequence of states $q_1, q_2, \ldots, q_r$ is a connecting sequence if

- $\forall a \in \Sigma$, $\delta(q_i, a)$ *and* $\delta(q_{i+1}, a)$ are on same list
- The pair (q_i, q_{i+1}) is on stack

Joined States

States p and q are joined by the connecting sequence $q_1, q_2, \ldots, q_r$ if $p = q_1$ and $q = q_r$

Lemma

Lemma

E is an equivalence relation defined on $p, q \in S_1 \bigcup S_2$ s.t. pEq iff p and q appear on same list at the end of the algorithm. It is coarsest right invariant equivalence identifying s_1 and s_2.

Lemma

Lemma

E is an equivalence relation defined on $p, q \in S_1 \bigcup S_2$ s.t. pEq iff p and q appear on same list at the end of the algorithm. It is coarsest right invariant equivalence identifying s_1 and s_2.

Proof :

- **Coarsest Equivalence Relation**
  Two lists are merged only if $\exists p_1, p_2 \in Q_1 \bigcup Q_2$ are on the same list and $\forall a \in \Sigma \delta_1(p_1, a)$ *and* $\delta(p_2, a)$ are on different lists. Since does not make too many identifications $\Rightarrow$ it is coarsest.

Lemma - Proof Contd.

- **Right Invariant Equivalence Relation**
  Induction Hypothesis : Before k^{th} iteration of the 'while' loop, if (p, q) are on the same list, then p and q are joined by a connecting sequence.
  Basis : k=1
  s_1, s_2 are only in the same set and (s_1, s_2) are at the stack top.
  $\Rightarrow$ s_1 *and* s_2 are joined by a connecting sequence.
  Induction Step :
  - If p and q are joined before the k^{th} iteration, they are joined after k^{th} iteration also
  - Assume p and q are on the same list after k^{th} iteration
    1. p and q were on same list before the k^{th} iteration, they remain so.
    2. Several lists merge into one list because the join relation is reflexive, symmetric and transitive.

Theorem

Lemma

The given algorithm is correct.

Theorem

Lemma

The given algorithm is correct.

Proof :

- $M_1 \equiv M_2$
  - Let E' be a right invariant equivalence relation s.t. $\forall p, q \in Q_1 \bigcup Q_2 \; \forall w \in \Sigma^* \; \hat{\delta}(p, w) \in F_1 \bigcup F_2$ *iff* $\hat{\delta}(q, w) \in F_1 \bigcup F_2$
  - Since E' is right invariant $\Rightarrow$ E' is a refinement of E
  - Since E' can not identify final and non-final states neither can E

    $\Rightarrow$ No list can contain both final and non-final state.

Theorem - Proof Contd.

- **If $M_1 \neq M_2$ some list contains final and non-final state**
  - $\exists w \in \Sigma^* : \hat{\delta}(s_1, w) \in F$ *and* $\hat{\delta}(s_2, w) \notin F$
  - Since E is right invariant(Lemma), $\hat{\delta}(s_1, w)$ E $\hat{\delta}(s_2, w)$
  - $\Longrightarrow \hat{\delta}(s_1, w)$ *and* $\hat{\delta}(s_2, w)$ are in the same list
  - $\Longrightarrow$ A list contains final and non-final state

Time Complexity Analysis

Theorem

Execution time of the algorithm is $n \times (|Q_1| + |Q_2|)$.

Time Complexity Analysis

Theorem

Execution time of the algorithm is $n \times (|Q_1| + |Q_2|)$.

Proof :

- Step 1, 2 and 3 take $O(n)$ time.
- Step 3 takes $O(m \times |\Sigma|)$ time where m is the number of pairs pushed/popped on the stack
  - Each time a pair is pushed on to the stack, total number of sets are decreased by 1.
  - As there were n sets in the beginning, atmost (n-1) pairs are pushed/ popped.
  - Number of pairs pushed/ popped from the stack is therefore bounded by n.

Questions??

Thank You!!