Theory of Computation

Unit 2: Regular Languages, DFAs and NFAs

Syedur Rahman

Lecturer, CSE Department
North South University
syedur.rahman@wolfson.oxon.org

Theory of Computation Lecture Notes

Acknowledgements

These lecture notes contain material from the following sources:

[Plump] D.J. Plump: *Theory of Computation*, Dept of Computer Science, University of York, 2003

[MOC] *Models of Computation*, Oxford University Computing Laboratory, University of Oxford, 2005.

[MCS] *Mathematics for Computer Scientists*, Dept of Computer Science, University of York, 2003

Preliminary Concepts

An *alphabet* is a finite set (denoted by $\Sigma, \Gamma, \ldots$) the elements of which are called *symbols*.

Examples: $\Sigma = \{0, 1\}$ and $\Gamma = \{a, b, \ldots, z\}$.

A *string* over an alphabet Σ is a finite sequence of symbols from Σ, written by juxtaposing symbols. The *empty string* is denoted by Λ (other authors use λ or ϵ).

Examples: 0111011 and $watermelon$ are strings over Σ and Γ, respectively.

From [Plump]

Preliminary Concepts (cntd.)

The *length* of a string w, denoted by $|w|$, is its length as a sequence.

Examples: $|watermelon| = 10$ and $|\Lambda| = 0$.

The *set of all strings* over an alphabet Σ is denoted by Σ^*. (Note that $\Lambda \in \Sigma^*$ and that Σ^* is infinite unless Σ is the empty set.)

A *formal language*, or *language* for short, over an alphabet Σ is a subset of Σ^*.

Examples:
$\{\Lambda, 0, 00, 001\}$ and $\{w \in \{0,1\}^* \mid |w| = 2^n \text{ for some } n \geq 0\}$ are languages over $\{0, 1\}$.

From [Plump]

Preliminary Concepts (cntd.)

Let $u, v \in \Sigma^*$. The *concatenation* of u and v, denoted by uv, is recursively (or *inductively*) defined as follows:

(1) $uv = u$ if $v = \Lambda$.

(2) $uv = (uw)a$ if $v = wa$ for some $w \in \Sigma^*$ and $a \in \Sigma$.

Concatenation of languages: For $L_1, L_2 \subseteq \Sigma^*$,

$$L_1L_2 = \{uv \mid u \in L_1 \text{ and } v \in L_2.\}$$

From [Plump]

Preliminary Concepts (cntd.)

Notation: For $a \in \Sigma$, $w \in \Sigma^*$, $L \subseteq \Sigma^*$ and $k \geq 1$,

$$\begin{aligned} a^k &= aa \cdots a \\ w^k &= ww \cdots w \\ L^k &= L \cdots L \end{aligned}$$

where in each case there are k factors on the right-hand side.

For $k = 0$: $a^0 = w^0 = \Lambda$ and $L^0 = \{\Lambda\}$.

Kleene star: For a language L,

$$L^* = \bigcup_{i=0}^{\infty} L^i \quad \text{and} \quad L^+ = \bigcup_{i=1}^{\infty} L^i.$$

From [Plump]

Regular Languages

The *regular languages* over an alphabet Σ are inductively defined as follows:

(1) The empty set $\emptyset$ and $\{a\}$, for each a in Σ, are regular languages over Σ.

(2) If L_1 and L_2 are regular languages over Σ, then

$$L_1 \cup L_2 \quad \text{and} \quad L_1 L_2 \quad \text{and} \quad L_1^*$$

are regular languages over Σ.

From [Plump]

Regular Languages and Operators

Let A and B be languages. Define

- **Union**: $A \cup B = \{ x : x \in A \text{ or } x \in B \}$
- **Concatenation**: $A \cdot B = \{ xy : x \in A \text{ and } y \in B \}$
- **Star**: $A^* = \{ x_1 x_2 \cdots x_k : k \geq 0 \text{ and each } x_i \in A \}$.

Note: ϵ (the empty string) is in A^* (the case of $k = 0$)

Example. Take $A = \{ good, bad \}$ and $B = \{ boy, girl \}$.

$A \cdot B = \{ goodboy, goodgirl, badboy, badgirl \}$

$A^* =$

$\{ \epsilon, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, \cdots \}$

Informally $A^* = \{ \epsilon \} \cup A \cup (A \cdot A) \cup (A \cdot A \cdot A) \cup \cdots$.

Regular Expressions

Regular languages can be specified by *regular expressions* which are inductively defined as follows:

(1) $\emptyset$ and $\mathbf{a}$, for each a in Σ, are regular expressions over Σ.

(2) If r_1 and r_2 are regular expressions over Σ, then

$$(r_1 + r_2) \quad \text{and} \quad (r_1 r_2) \quad \text{and} \quad (r_1^*)$$

are regular expressions over Σ.

Examples of Regular Expressions

Given **Σ = {a, b}.** The following are regular expressions specifying languages over Σ.

ab + ba

a*b*

(ba)* (ba)* + a*b*

(ba)*(ab + bba + Λ)

Examples of Regular Expressions

Given **Σ = {a, b}.** The following are regular expressions specifying languages over Σ.

ab + ba specifies a language that only contains ab and ba.

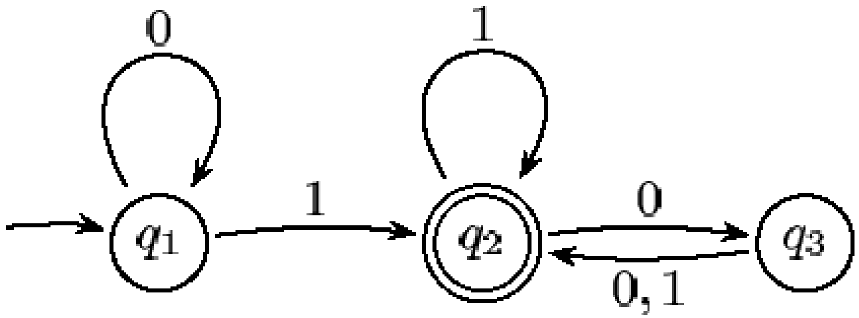
a*b* specifies a language that contains only strings of any number of (or 0) a's followed by any number of (or 0) b's.

(ba)* specifies a language that contains only strings containing a number of (or 0) repetitions of ba.

(ba)* + a*b* specifies a language that contains a string iff the string is contained by either of the two previous languages.

(ba)*(ab + bba + Λ) specifies a language that contains strings starting with a number of (or 0) repetitions of ba followed by ab, bba or nothing.

Deterministic Finite State Automata
An Example of a DFA:



Key Features:

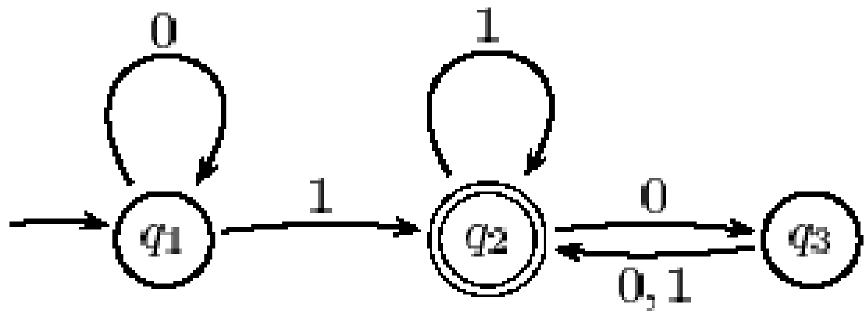
- There are only finitely different *states* a finite automaton can be in.

  The states in M_1 (= vertices of the graph) are q_1, q_2 and q_3.

- We do not care about the internal structure of automaton states. All we care about is which *transitions* the automaton can make between states.

- A symbol from some finite alphabet Σ is associated with each transition: we think of elements of Σ as *input symbols*. The alphabet of M_1 is $\{ 0, 1 \}$.

From [MOC]



- Thus all possible transitions can be specified by a *finite directed graph with Σ-labelled edges*.

  E.g. At state q_2, M_1 can

  - input 0 and enter state q_3 i.e. $q_2 \xrightarrow{0} q_3$, or
  - input 1 and remain in state q_2 i.e. $q_2 \xrightarrow{1} q_2$.

- There is a distinguished *start state*. In the graph, the start state is indicated by an arrow pointing at it from nowhere. The start state of M_1 is q_1.

- The states are partitioned into *accepting* states (or final states) and *non-accepting* states.

  An accepting state is indicated by a (double) circle. The accepting state of M_1 is q_2.

From [MOC]

Formal Definition of a DFA

A *deterministic finite automaton (DFA)* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

(i) Q is a finite set called the *states*

(ii) Σ is a finite set called the *alphabet*

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

(iv) $q_0 \in Q$ is the *start state*

(v) $F \subseteq Q$ is the *set of accept states* (or *final states*).

We write $q \xrightarrow{a} q'$ to mean $\delta(q, a) = q'$, which we read as "there is an a-*transition* from q to q'".

From [MOC]

Languages accepted by a DFA

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. $L(M)$, the *language recognized (or accepted) by the DFA* M, consists of all strings $w = a_1 a_2 \cdots a_n$ over Σ satisfying $q_0 \xrightarrow{w}{}^{*} q$ where q is a final state. Here

$$q_0 \xrightarrow{w}{}^{*} q$$

means that there exist states $q_1, \cdots, q_{n-1}, q_n = q$ (not necessarily all distinct) such that there are transitions of the form

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n = q$$

Note

- case $n = 0$: $q \xrightarrow{\epsilon}{}^{*} q'$ iff $q = q'$
- case $n = 1$: $q \xrightarrow{a}{}^{*} q'$ iff $q \xrightarrow{a} q'$

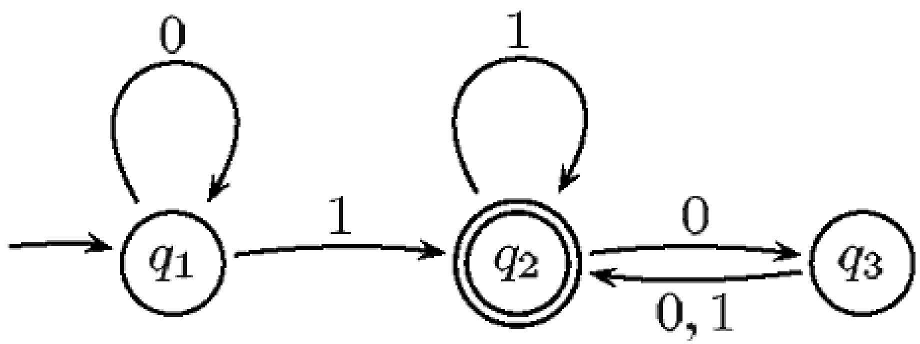
A language is called *regular* if some DFA recognizes it.

From [MOC]

Definition of DFA: An Example

Formally $M_1 = (Q, \Sigma, \delta, q_1, F)$ where

- $Q = \{ q_1, q_2, q_3 \}$
- $\Sigma = \{ 0, 1 \}$
- q_1 is the start state; $F = \{ q_2 \}$
- δ is given by



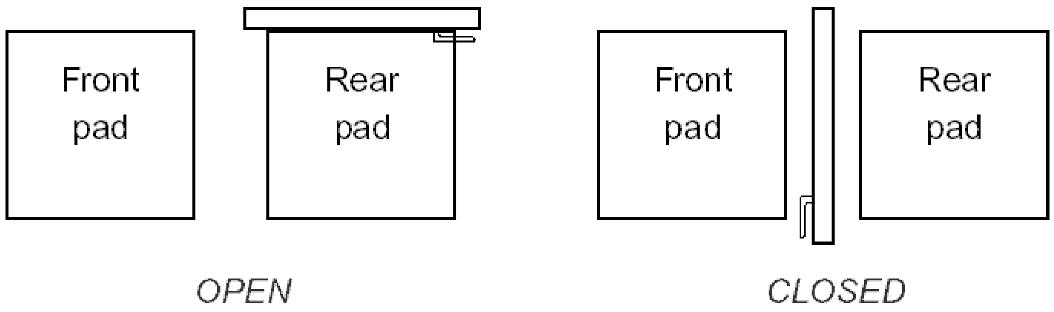
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

$L(M_1)$ is the set of all binary strings that contain at least one 1, and an even number of 0s follow the last 1.

From [MOC]

DFA: A Practical Example

An Automatic Door Controller



States Q = {OPEN, CLOSED}

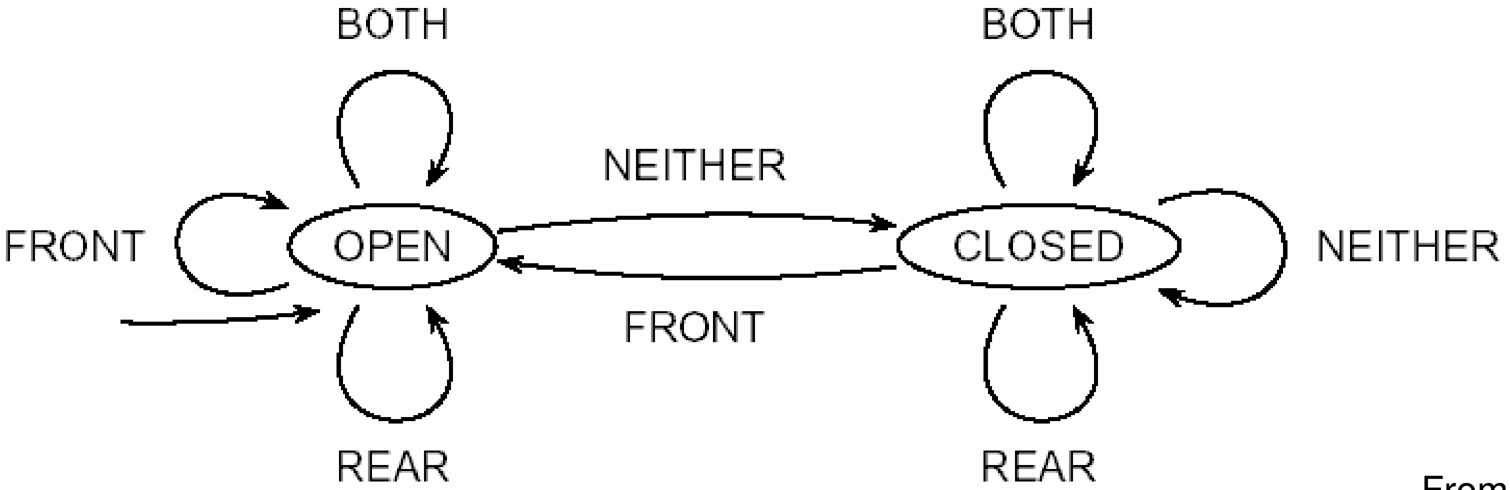
Alphabet Σ = {FRONT, REAR, BOTH, NEITHER}
FRONT: someone standing on front pad only
REAR: someone standing on rear pad only
BOTH: people standing on both pads
NEITHER: no one standing on either pad

DFA: A Practical Example

State transition table δ:

	NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

Start State q_o: OPEN



From [MOC]

Construct DFAs from the following expressions

Given **Σ = {a, b}.** The following are regular expressions specifying languages over Σ.

ab + ba specifies a language that only contains ab and ba.

a*b* specifies a language that contains only strings of any number of (or 0) a's followed by any number of (or 0) b's.

(ba)* specifies a language that contains only strings containing a number of (or 0) repetitions of ba.

(ba)* + a*b* specifies a language that contains a string iff the string is contained by either of the two previous languages.

(ba)*(ab + bba + Λ) specifies a language that contains strings starting with a number of (or 0) repetitions of ba followed by ab, bba or nothing.

Regular Languages and Expressions

language	regular expression
$\{a\}$	a
$\{a, b\}$	$a + b$
$\{a, b\} \cup \{b, c\}$	$(a + b) + (b + c)$
$\{a, b\}^* \cup \{aa, bc\}$	$(a + b)^* + (aa + bc)$
$\{a, b\}^* \cdot \{ab, ba\}^*$	$(a + b)^*(ab + ba)^*$
$\{a, b\}^* \cup \{\lambda, aa\}^*$	$(a + b)^* + (\lambda + aa)^*$

+ is union; . is concatenation; * star closure

From [MOC]

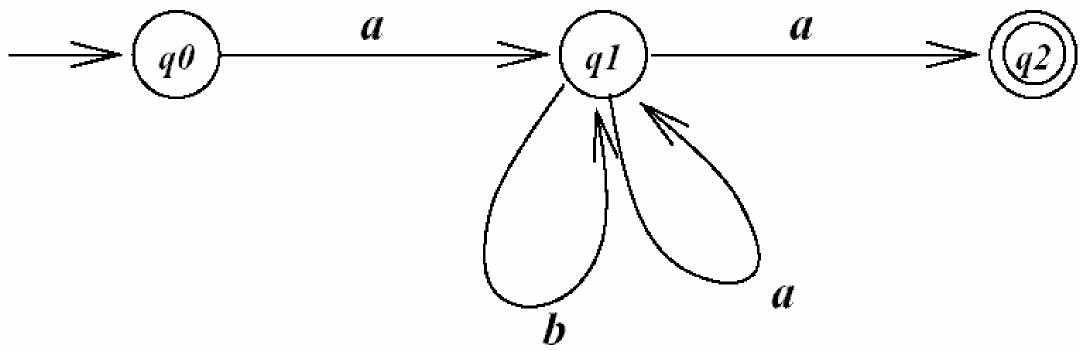
Nondeterministic Finite State Automata (NFA)

A *non-deterministic* finite state automaton (NFA) is specified by a tuple, $M = < Q, \Sigma, \rho, i, F >$ where

- Q, a finite set of states
- Σ, a finite set of possible input symbols, the *alphabet*
- ρ, a transition **relation** $Q \times (\Sigma \cup \{\lambda\}) \times Q$
- i in Q, an initial state
- F, a set of final states ($F \subseteq Q$)

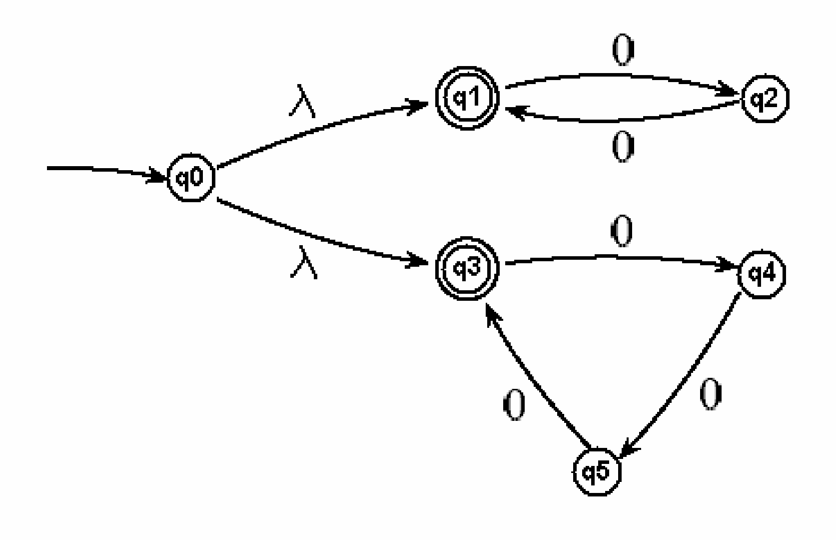
From [MCS]

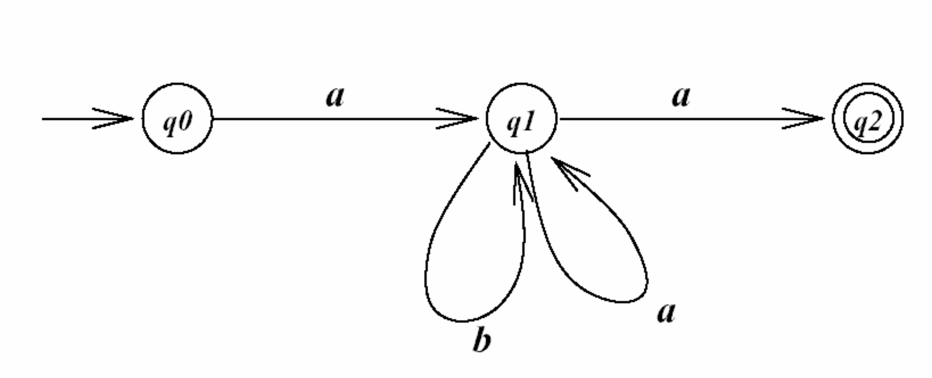
Example of an NFA

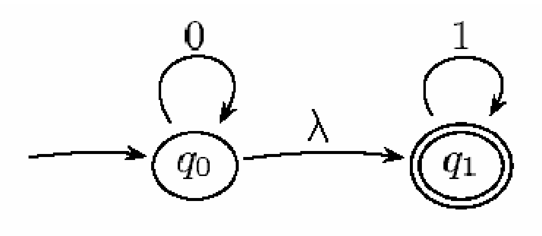


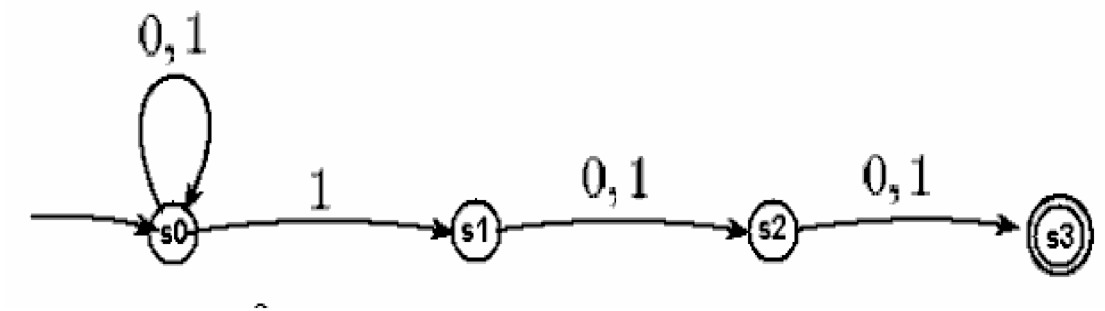
From [MCS]

Examples of NFAs



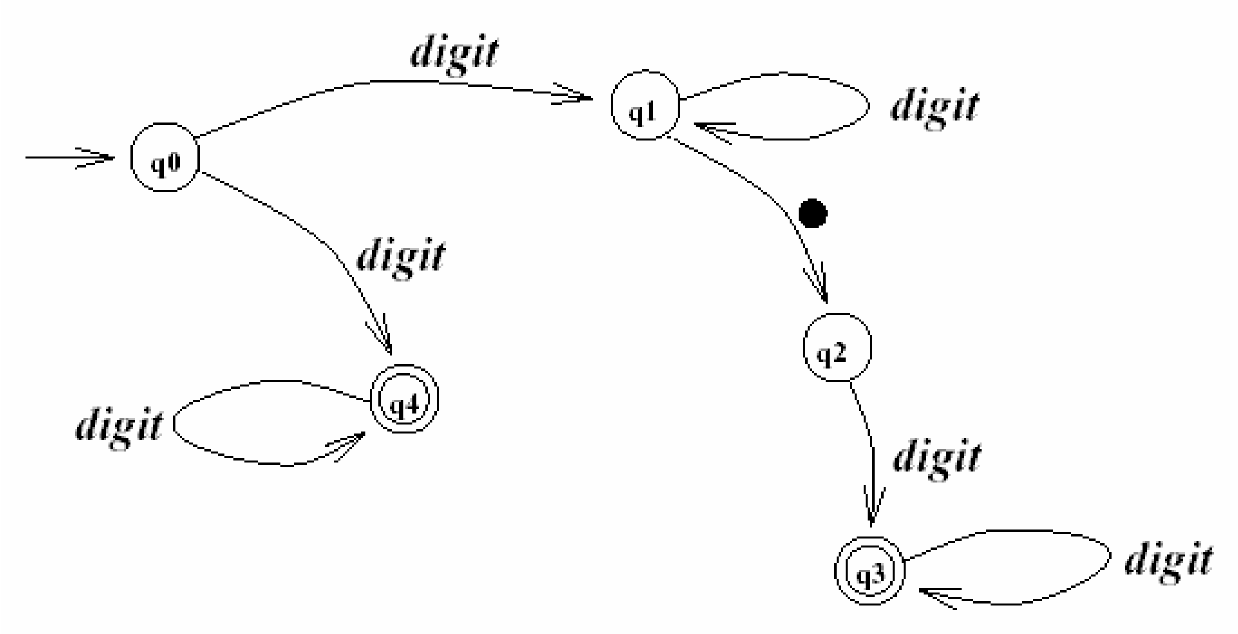






From [MCS]

Example of an NFA



From [MCS]

DFA vs NFA

- In a DFA, at every state q, for every symbol a, there is a unique a-transition i.e. there is a unique q' such that $q \xrightarrow{a} q'$.

  This is not necessarily so in an NFA. At any state, an NFA may have multiple a-transitions, or none.

- In a DFA, transition arrows are labelled by symbols from Σ; in an NFA, they are labelled by symbols from $\Sigma \cup \{\lambda\}$. I.e. an NFA may have λ-transitions.

- We may think of the non-determinism as a kind of parallel computation wherein several processes can be running concurrently.

  When the NFA splits to follow several choices, that corresponds to a process "forking" into several children, each proceeding separately. If at least one of these accepts, then the entire computation accepts.

From [MCS]

Every NFA has an equivalent DFA

Observation. Every DFA *is* an NFA!

Say two automata are *equivalent* if they accept the same language.

Theorem(Determinization). Every NFA has an equivalent DFA.

Proof. Fix an NFA $N = (Q_N, \Sigma_N, \delta_N, q_N, F_N)$, we construct an equivalent DFA $\mathcal{P}N = (Q_{\mathcal{P}N}, \Sigma_{\mathcal{P}N}, \delta_{\mathcal{P}N}, q_{\mathcal{P}N}, F_{\mathcal{P}N})$ such that $L(N) = L(\mathcal{P}N)$:

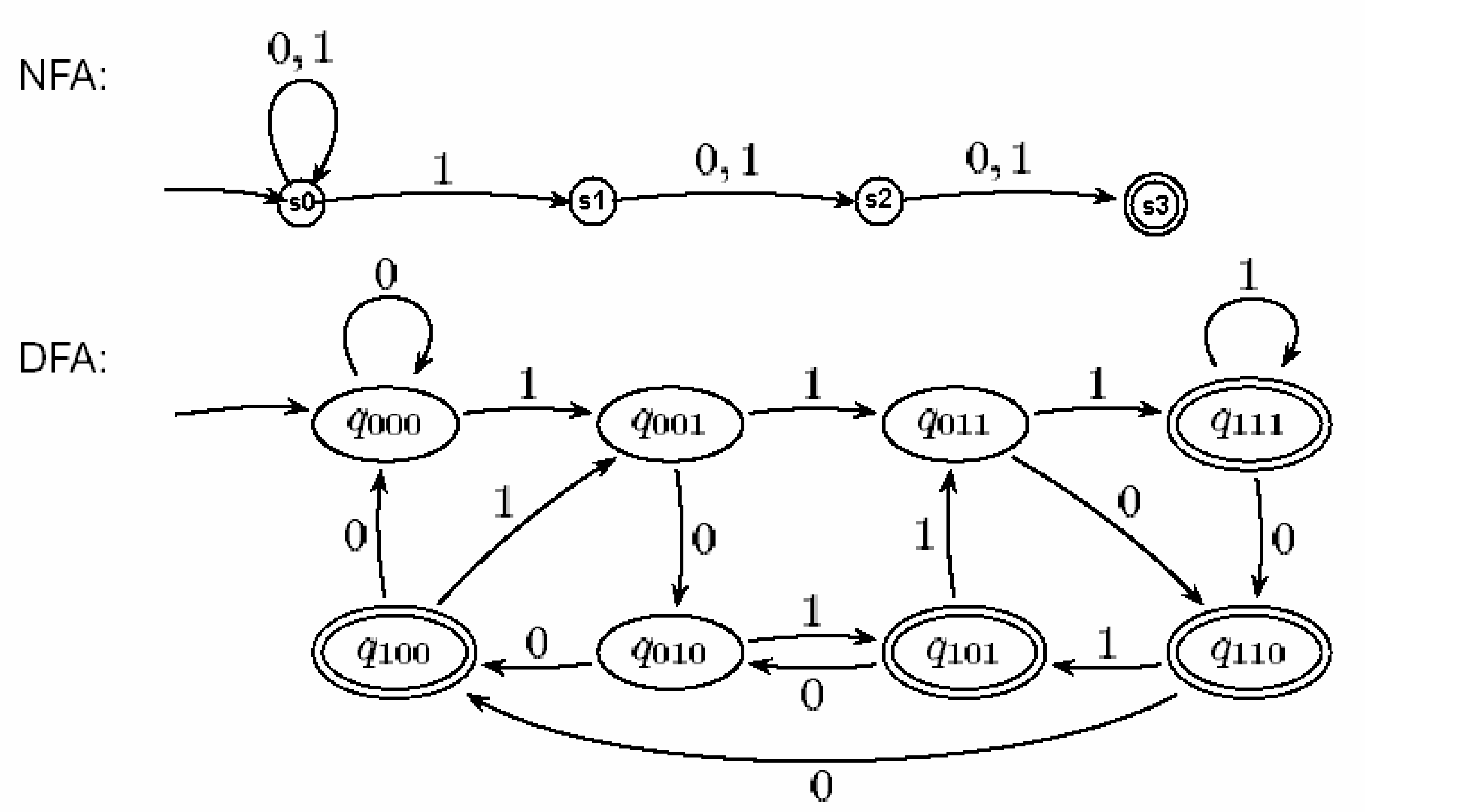
- $Q_{\mathcal{P}N} \stackrel{\text{def}}{=} \{ S : S \subseteq Q_N \}$
- $\Sigma_{\mathcal{P}N} \stackrel{\text{def}}{=} \Sigma_N$
- $S \xrightarrow{a} S'$ in $\mathcal{P}N$ iff $S' = \{ q' : \exists q \in S.(q \stackrel{a}{\Longrightarrow} q' \text{ in } N) \}$
- $q_{\mathcal{P}N} \stackrel{\text{def}}{=} \{ q : q_N \stackrel{\epsilon}{\Longrightarrow} q \}$
- $F_{\mathcal{P}N} \stackrel{\text{def}}{=} \{ S \in Q_{\mathcal{P}N} : F_N \cap S \neq \emptyset \}$

From [Plump]

NFAs are often used as simpler representations of DFAs

Remember that all DFAs are NFAs as well but not all NFAs are DFAs

Example: All strings containing 1 in the third position from the end



From [Plump]

Converting any NFA to a DFA

Step 1 In the first instance write down the individual transitions as separate labelled arrows between states

Step 2
- The start state for the new DFA is labelled $\{q_0\}$
- For each input symbol identify all the transitions that start at q_0. Collect all the resultant states and put them in a set.

  In the above example the only transitions that start at q_0 is $q_0 \xrightarrow{a} q_1$. Hence we get $\{q_0\} \xrightarrow{a} \{q_1\}$.
- For each new state $\{n_1, n_2, \ldots, n_k\}$ **repeat:**
  - For each input symbol a **repeat**
    - Collect all output states for each transition $n_i \xrightarrow{a} m_i \quad (1 \leq i \leq k)$ into a new state

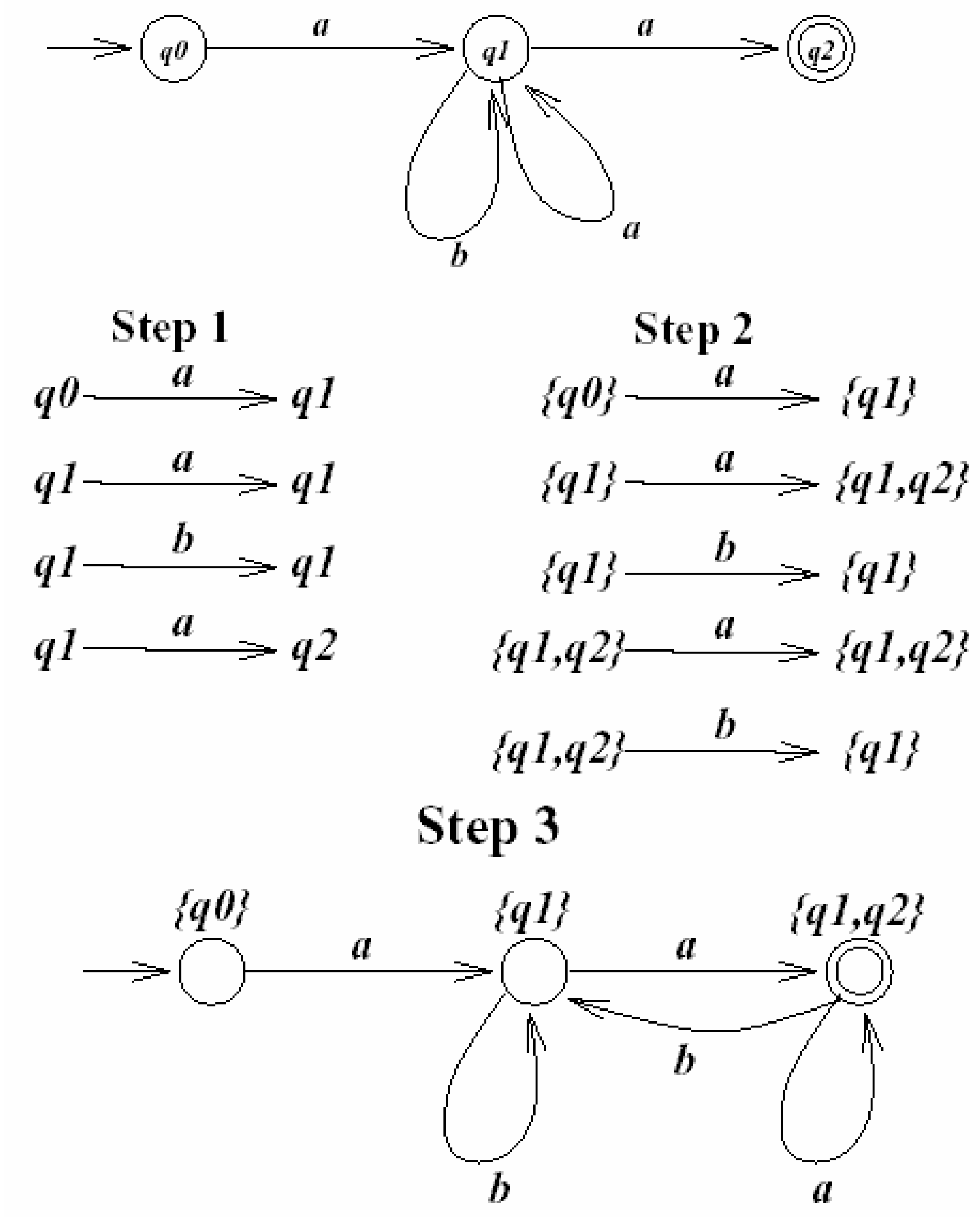
  This process is repeated until no new states are generated.
  (In the above example, the initial new state is $\{q_1\}$.
  We get $\{q_1\} \xrightarrow{a} \{q_1, q_2\}$ and $\{q_1\} \xrightarrow{b} \{q_1\}$ since there is no b-edge from q_1 to q_2.
  The only new state resulting from the above process is $\{q_1, q_2\}$.
  We get $\{q_1, q_2\} \xrightarrow{a} \{q_1, q_2\}$ and $\{q_1, q_2\} \xrightarrow{b} \{q_1\}$.
  No additional new states are generated and the process terminates.)

Step 3 You can now draw the transition diagram for the new DFA

Step 4 The final states in the new DFA are the states Q which contain an element $q \in Q$ such that q is a final state in the original NFA.

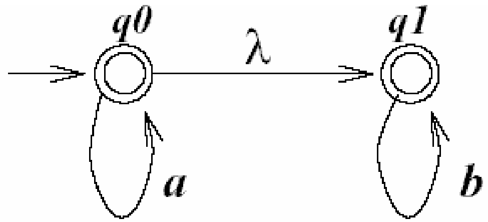
From [MCS]

Converting an NFA to a DFA

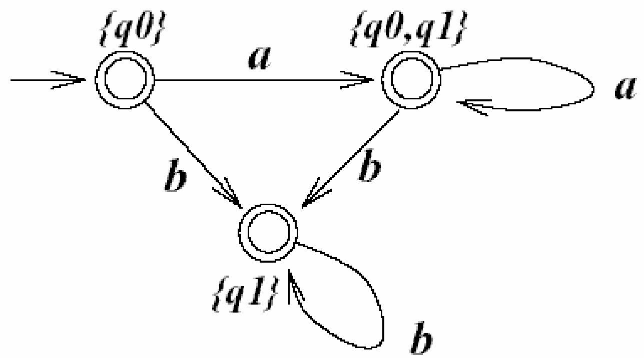


From [MCS]

Converting an NFA to a DFA



$q_0 \xrightarrow{a} q_0$	$\{q_0\} \xrightarrow{a} \{q_0, q_1\}$
$q_0 \xrightarrow{a} q_1$	$\{q_0\} \xrightarrow{b} \{q_1\}$
$q_0 \xrightarrow{b} q_1$	$\{q_1\} \xrightarrow{b} \{q_1\}$
$q_1 \xrightarrow{b} q_1$	$\{q_0, q_1\} \xrightarrow{a} \{q_0, q_1\}$
	$\{q_0, q_1\} \xrightarrow{b} \{q_1\}$



From [MCS]

Regular Languages and FSAs

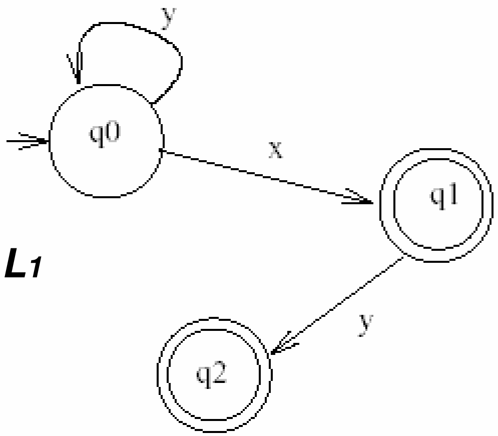
A language is regular if and only if a finite state automata (deterministic or non-deterministic) can be constructed that accepts it.

If two languages L_1 and L_2 are regular, the following languages are regular as well:

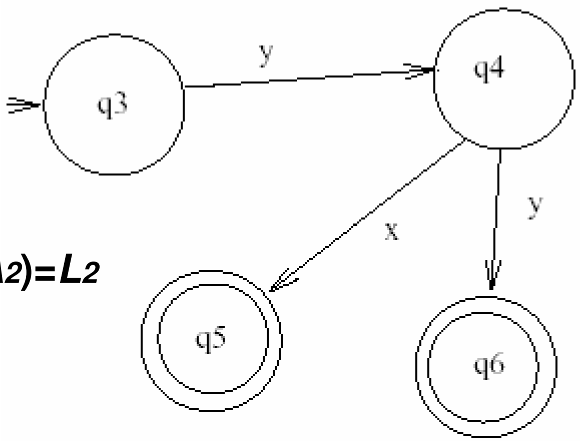
$L_1 \cup L_2$	$L_1 . L_2$	$L_1 . L_1$	$L_1 . L_1 . L_1$	
L_1^n	L_1^*	$L_1 \cap L_2$	$L_1 - L_2$	$\overline{L_1}$

This can be proved by showing how each of the given languages are accepted by finite state automata (NFAs or DFAs) via construction.

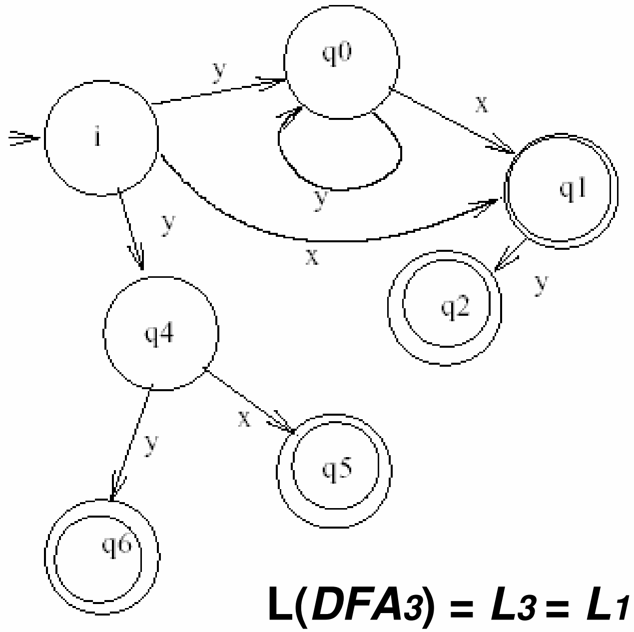
Union of Languages, $L_1 \cup L_2$



L(*DFA1*)=*L1*



L(*DFA2*)=*L2*



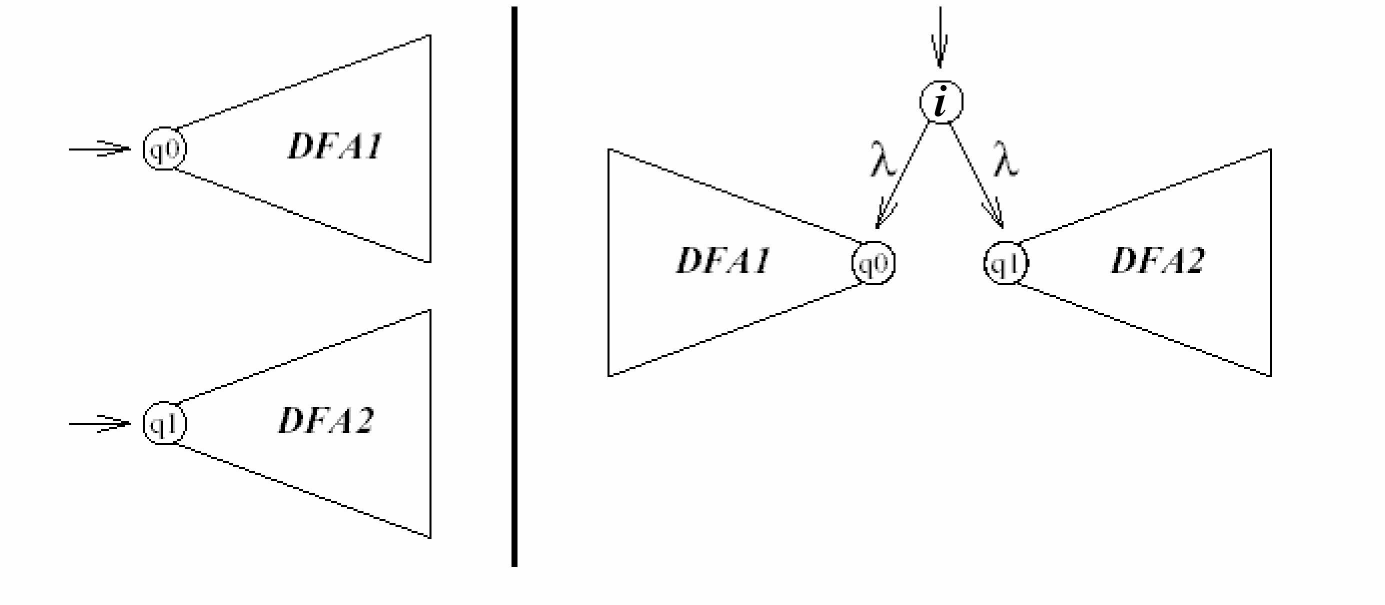
L(*DFA3*) = *L3* = *L1* U *L2*

- So we can always construct a new initial state and make the arcs from the individual initial states leave this new state and go to the states of the distinct machines. In the event that the previous initial states contain no back loops we can remove the previous initial states.

From [MCS]

$L_1 \cup L_2$ using an NFA

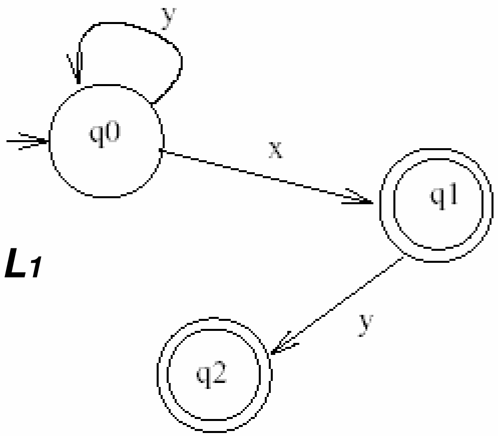
We can create a new NFA that accepts L(*DFA1*) U L(*DFA2*) by adding a new start state *i* and adding transitions from *i* to the start states of *DFA1* and *DFA2*



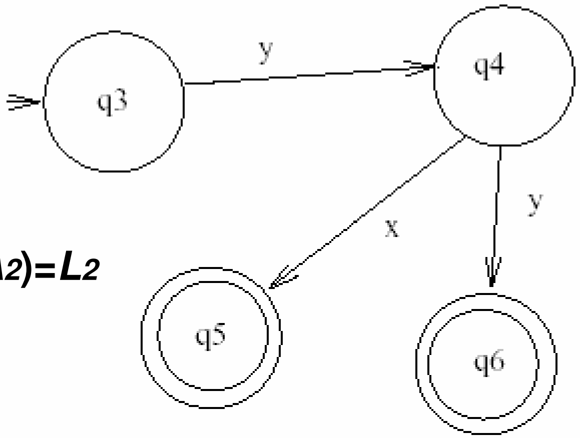
From [MCS]

Concatenation of Languages, L1.L2

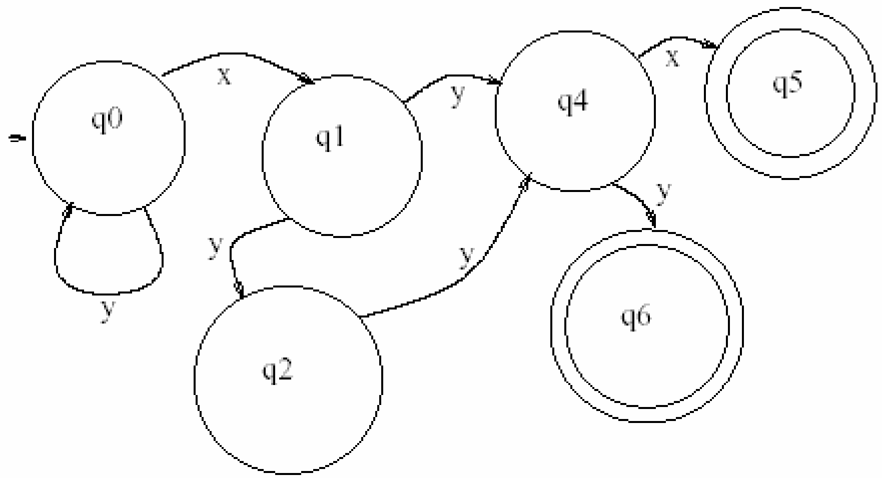
L(*DFA1*)=*L1*



L(*DFA2*)=*L2*



L(*DFA3*) = *L3* = *L1.L2*

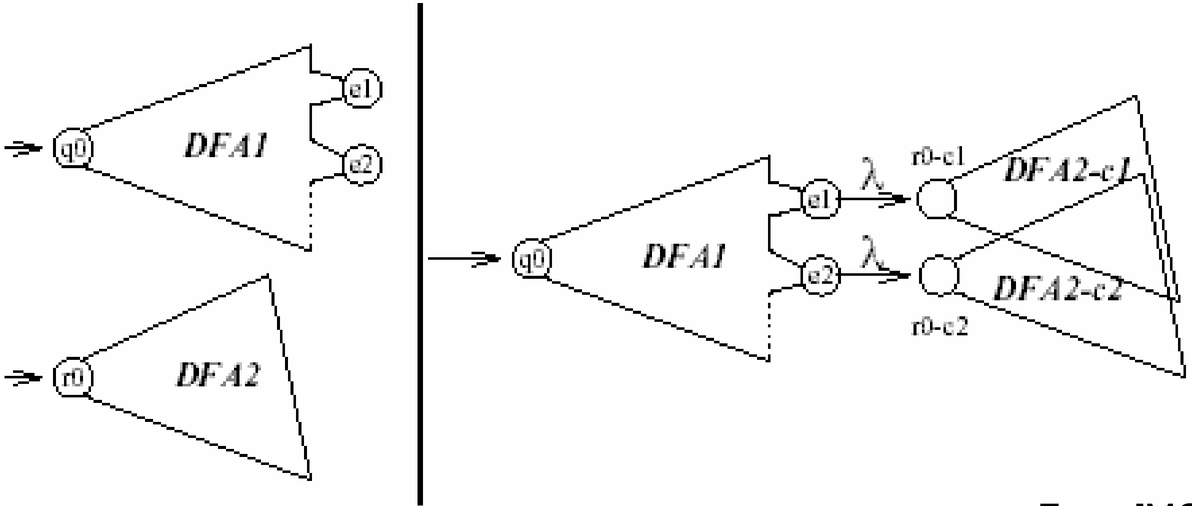


- From each accept state of the first DFA draw an arc to each state of the second that is the destination of its initial state. Allow the accept states of the second DFA to continue to be accept states and let accept states of the first DFA to be accept states only if the initial state of the second is an accept state.

From [MCS]

$L_1.L_2$
using an NFA

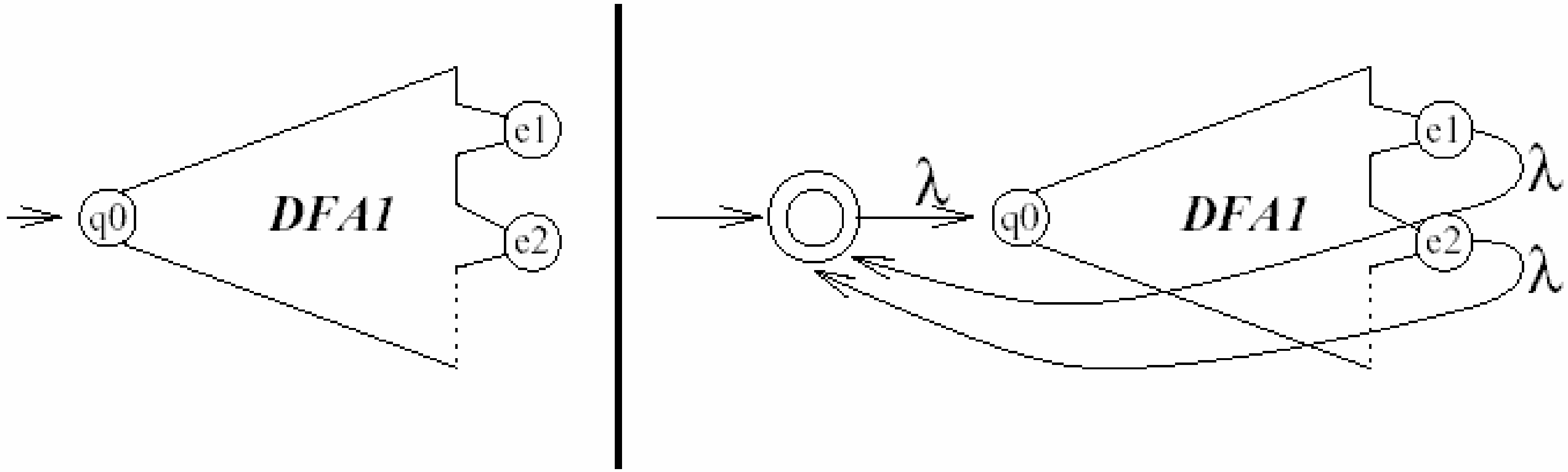
- In the general case DFA_1 is going to have several final states
- Assume that DFA_1 has n final states
- Make n copies of DFA_2
- We can make copies of DFA_2 by consistently giving new names to each state in DFA_2
- For example we can rename the start state r_0 to $r_0 - c1$ in the copy
- Repeat the above procedure to connect each final state of DFA_1 and the initial state of a copy of DFA_2 with a λ-transition as shown below



From [MCS]

Star closure of a language, L1 using an NFA*

- In the general case DFA_1 is going to have several final states
- Assume that DFA_1 has n final states
- We connect all the final states and the initial state of DFA_1 with λ-transitions as shown in the diagram below



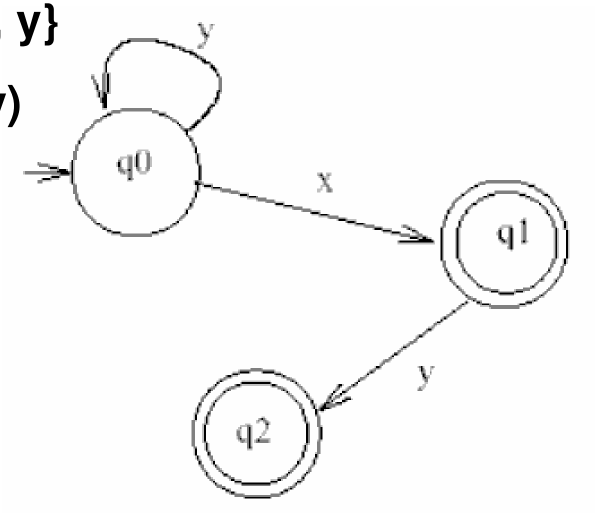
From [MCS]

Negation of a Language, $\overline{L_1}$

Σ = {x, y}

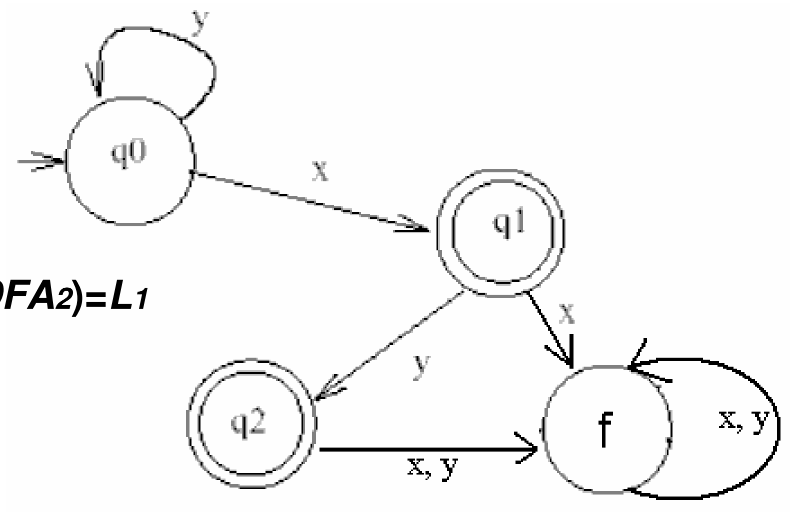
$L_1 = \{y\}^*.x.\{\lambda, y\}$

$E_1 = y^*x(\lambda + y)$



L(*DFA1*)=*L1*

Notice that all DFA's have implicit transitions to a failure or rejection state, if there is no transition mentioned for a state for a particular character of the alphabet.

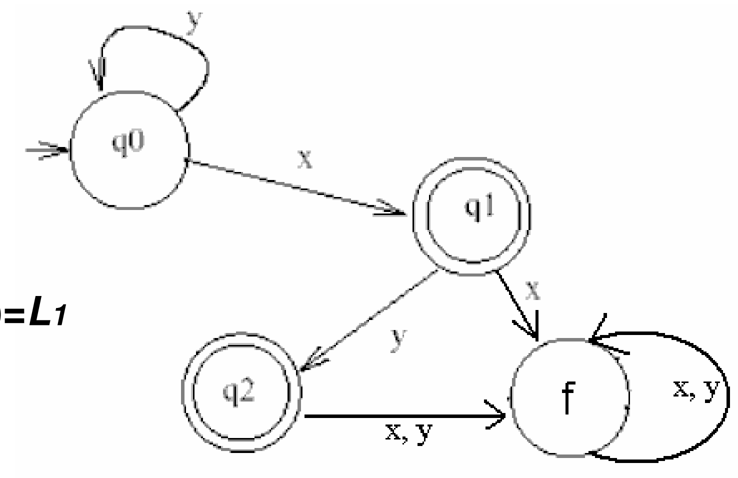


L(*DFA2*)=*L1*

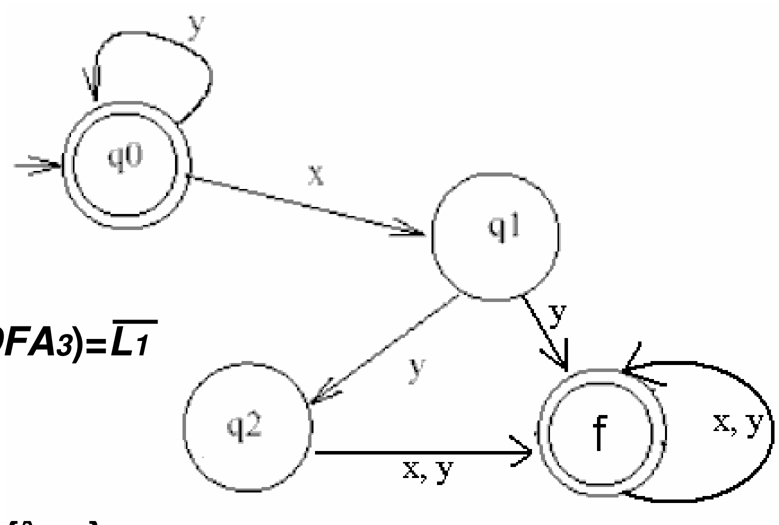
So for any DFA1, one could construct DFA2 which includes explicitly mentions all the missing transitions leading to a failure/rejection state f. Transitions for all symbols from f must lead back to f.

From [MCS]

Negation of a Language, $\overline{L_1}$



L(*DFA2*)=*L1*



L(*DFA3*)=$\overline{L_1}$

Σ = {x, y}

L_1 = {y}*.x.{λ, y}

- For *DFA1*, construct *DFA2* which includes explicitly mentions all the missing transitions leading to a failure state *f*.
- Construct *DFA3*, which is copy of *DFA2* with all the accepting states in *DFA2* as non-accepting states in *DFA3* and all the other states in *DFA2* as accepting states in *DFA3*. Make sure that *f* is an acceptance state.
- *DFA3* now accepts the complement of *L1*

From [MCS]