Chapter 3

Nondeterministic Finite Automata (NFA)

Nondeterminism

- An important notions(or abstraction) in computer science
- refer to situations in which the next state of a computation is not uniquely determined by the current state.
  - Ex: find a program to compute max(x,y):
  - pr1: case $x \geq y$ => print x;
  - $y \geq x$ => print y
  - endcase;
  - Then which branch will be executed when x = y ?
  - ==> *don't care nondeterminism*
  - Pr2: do-one-of {
  - {if x < y fail; print x},
  - {if y < x fail, print y} }.
  - ==>The program is powerful in that it will never choose branches that finally lead to ‘fail’ -- an unrealistic model.
  - ==> *don't know nondeterminism*.

nondeterminism (cont'd)

- a nondeterministic sorting algorithm:
- nondet-sort(A, n)
  - 1. for i = 1 to n do
  - 2. nondeterministically let k := one of {i, ..., n} ;
  - 3. exchange A[i] and A[k]
  - 4. endfor
  - 5 for i = 1 to n-1 do if A[i] > A[i+1] then fail;
  - 6. return(A).
  - Notes: 1. Step 2 is magic in that it may produce many possible outcomes. However all incorrect results will be filtered out at step 5.
  - 2. The program run in time NTIME O(n)
  - cf: O(n lg n) is required for all sequential machines.

nondeterminism (cont'd)

- **Causes of nodeterminism in real life:**
  - **incomplete information about the state**
  - **external forces affecting the course of the computation**
  - **ex: the behavior of a process in a distributed system**
- **Nondeterministic programs *cannot be executed directly but can be simulated by real machine*.**
- **Nondeterminism can be used as a tool for the specification of problem solutions.**
- **an important tool in the design of efficient algorithms**
  - **There are many problems with efficient nondeterministic algorithm but no known efficient deterministic one.**
  - **the open problem NP = P ?**
- **How to make DFAs become nondeterministic ?**

==> allow multiple transitions for each state-input-symbol pair

==> modify the transition function δ.

Formal Definition of NFAs

- A NFA is a five-tuple $N = (Q,\Sigma,\delta,S,F)$ where everything is the same as in a DFA, except:
  - $S \subseteq Q$ is a set of *starting states*, instead of a single state.
  - δ is the *transition function* $\delta: Q \times \Sigma \to 2^Q$. For each state p and symbol a, $\delta(p,a)$ is the set of all states that N is allowed to move from p in one step under input symbol a.
  - diagrammatic notation: p --a--> q

    Note: $\delta(p,a)$ can be the empty set

$$p \xrightarrow{a} q$$

- The extended transition function Δ (multi-step version of δ) for NFA can be defined analogously to that of DFAs:

$\Delta: 2^Q \times \Sigma^* \to 2^Q$ is defined inductively as follows:

1. Basis: $\Delta(A, \varepsilon) = \underline{\ A\ \ \ }$ for every set of states A (6.1)
2. Ind. case: $\Delta(A, xa) = \underline{\ \bigcup_{q \in \Delta(A,\ xa)} \delta(q,a)\ }$ for all $x \in \Sigma^*, a \in \Sigma$ (6.2)

Note: Intuitively $q \in \Delta(A,x)$ means q can be reached from some state of A after scanning input string x.

Languages accepted by NFAs

- Note: Like DFAs, the extended transition function Δ on a NFA N is uniquely determined by N.
  - pf: left as an exercise.
- $N = (Q,\Sigma,\delta,S,F)$: a NFA;   x: any string over Σ;

  Δ: the extended transition function of N.

1. x is said to be *accepted* by N if $\Delta(S,x) \cap F \neq \{\}$
   - i.e., x is accepted if there is an accept state $q \in F$ such that q is reachable from a start state under input string x (i.e., $q \in \Delta(S,x)$)
2. The set (or language) accepted by N, denoted L(N), is the set of all strings accepted by N. i.e.,
   - $L(N) =_{def} \{x \in \Sigma^* \mid N \text{ accepts } x\}$.

Equivalence of FAs

- Two finite automata (FAs, no matter deterministic or nondeterministic) M and N are said to be equivalent if L(M) = L(N).

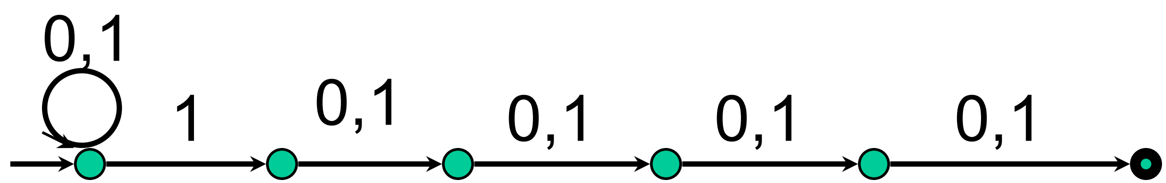
- Under such definition, every DFA $M = (Q,\Sigma,\delta,s,F)$ is equivalent to an NFA $N = (Q,\Sigma,\delta',\{s\},F)$ where
  - $\delta'(p,a) = \{\delta(p,a)\}$ for every state p and input a.

- Problem: Does the converse hold as well ?
  - i.e. For every NFA N there is a DFA M s.t. L(M) = L(N).
  - Ans: ____

Some examples of NFAs

Ex: Find a NFA accepting A = { $x \in \{0,1\}^*$ | the fifth symbol counted from the right is 1 } = {010000, 11111,...}.

Sol: 1. (in diagram form)

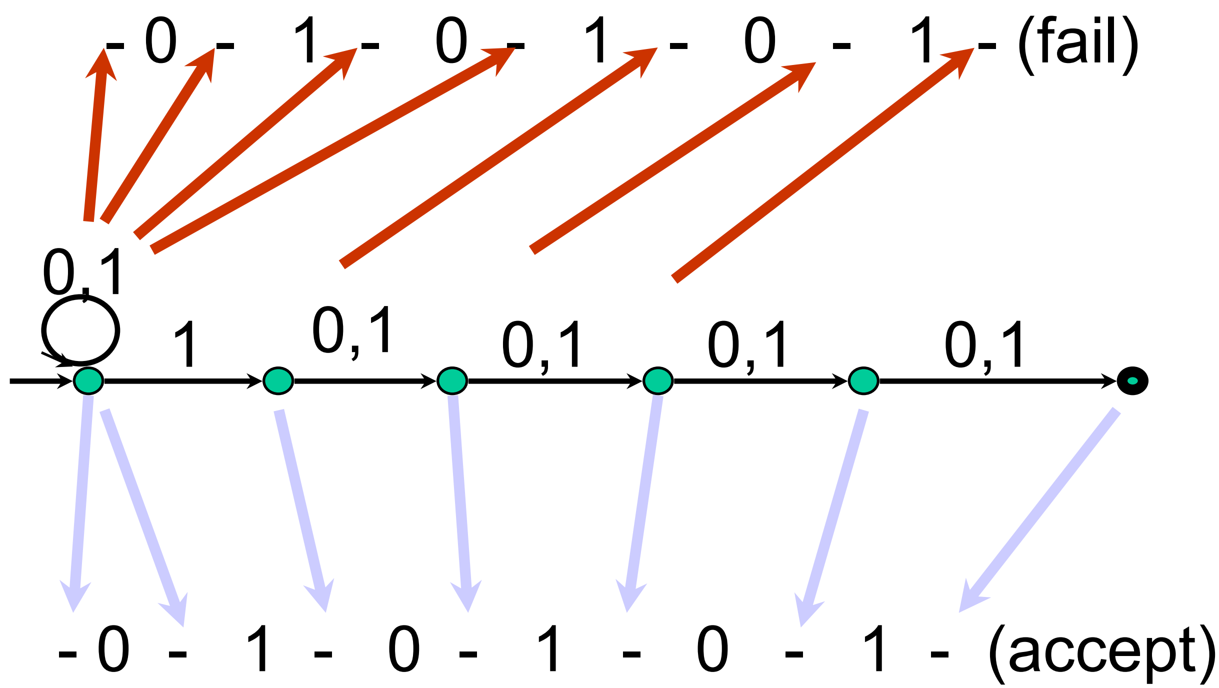


2: tabular form:

3. tuple form: (Q,Σ,δ,S,F) = (__,__,__,__,__).

Example of strings accepted by NFAs

- Note: there are many possible computations on the input string: 010101, some of which reach the (only) final state (accepted or successful computation), some of which do not (fail).
- Since there exists an accepted computation, by definition, the string is accepted by the machine



Some properties about the extended transition function Δ

- Lem 6.1: $\Delta(A,xy) = \Delta(\Delta(A,x),y)$.
- pf: by induciton on $|y|$:

  1. $|y| = 0 \Rightarrow \Delta(A,x\varepsilon) = \Delta(A,x) = \Delta(\Delta(A,x),\varepsilon)$ -- (6.1).
  2. $y = zc \Rightarrow \Delta(A,xzc) = \bigcup_{q \in \Delta(A,xz)} \delta(q,c)$ -- (6.2)

     $= \bigcup_{q \in \Delta(\Delta(A,x),z)} \delta(q,c)$ -- ind. hyp.

     $= \Delta(\Delta(A,x),zc)$ -- (6.2)

- Lem 6.2 Δ commutes with set union:
  - i.e., $\Delta(\bigcup_{i \in I} A_i,x) = \bigcup_{i \in I} \Delta(A_i,x)$. in particular, $\Delta(A,x) = \bigcup_{p \in A} \Delta(\{p\},x)$
- pf: by ind. on $|x|$. Let $B = \bigcup_{i \in I} A_i$

  1. $|x| = 0 \Rightarrow \Delta(\bigcup_{i \in I} A_i, \varepsilon) = \bigcup_{i \in I} A_i = \bigcup_{i \in I} \Delta(A_i, \varepsilon)$ -- (6.1)
  2. $x = ya \Rightarrow \Delta(\bigcup_{i \in I} A_i, ya) = \bigcup_{p \in \Delta(B,y)} \delta(p,a)$ -- (6.2)

     $= \bigcup_{p \in (\bigcup_{i \in I} \Delta(Ai,y))} \delta(p,a)$ -- ind. hyp. $= \bigcup_{i \in I} \bigcup_{p \in \Delta(Ai,x)} \delta(p,a)$ -- set theory $= \bigcup_{i \in I} \Delta(A_i,ya)$ (6.2)

The subset construction

- $N = (Q_N,\Sigma,\delta_N,S_N,F_N)$: a NFA.
- $M = (Q_M,\Sigma,\delta_M,s_M,F_M)$ (denoted 2^N): a DFA where
  - $Q_M = 2^{Q_N}$
  - $\delta_M(A,a) = \Delta_N(A,a)$ ($= \bigcup_{q\in A} \delta_N(q,a)$) for every $A \subseteq Q_N$.
  - $s_M = S_N$ and
  - $F_M = \{A \subseteq Q_N \mid A\cap F_N \neq \{\}\}$.
  - note: States of M are subsets of states of N.
- Lem 6.3: for any $A \subseteq Q_N$. and x in Σ^*, $\Delta_M(A,x) = \Delta_N(A,x)$.

pf: by ind on |x|. if $x = \varepsilon \Rightarrow \Delta_M(A,\varepsilon) = A = \Delta_N(A,\varepsilon)$. --(def)

if $x = ya \Rightarrow \Delta_M(A,ya) = \delta_M(\Delta_M(A,y),a)$ -- (def) $= \delta_M(\Delta_N(A,y),a)$ -- ind. hyp. $= \Delta_N(\Delta_N(A,y),a)$ -- def of δ_M $=\Delta_N(A, ya)$ -- lem 6.1

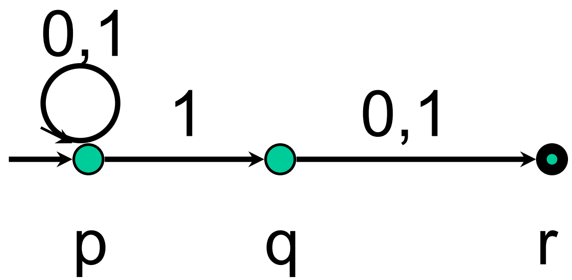
Theorem 6.4: M and N accept the same set.

pf: $x \in L(M)$ iff $\Delta_M(s_M,x)\in F_M$ iff $\Delta_N(S_N,x)\cap F_N \neq \{\}$ iff $x \in L(N)$.

Equivalence of NFAs and DFAs - an example

1. NFA N accepting A = { $x \in \{0,1\}^*$ | the second symbol from the right is 1 } = $\{x1a \mid x \in \{0,1\}^*$ and $a \in \{0,1\}\}$.

sol:



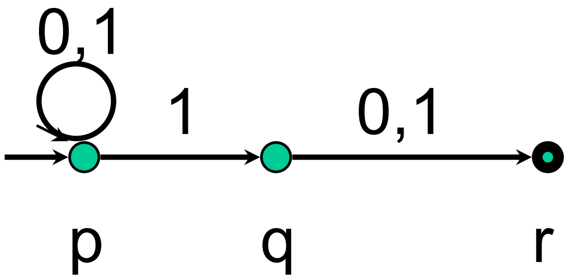
2. DFA M equivalent to N is given as :

		0	1
	{}	{}	{}
->	{p}	{p}	{p,q}
	{q}	{r}	{r}
	{r}F	{}	{}
	{p,q}	{p,r}	{p,q,r}
	{p,r}F	{p}	{p,q}
	{q,r}F	{r}	{r}
	{p,q,r}F	{p,r}	{p,q,r}

3. some states of M are redundant in the sense that they are never reachable from the start state and hence can be removed from the machine w/o affecting the languages accepted.

A more human friendly method

sol:



		0	1
0.	-> {p}	{p}	{p,q}
	{q}	{r}	{r}
	{r}F	{}	{}
1.	{p}	{p}	{p,q}
3.	{p,q}	{p,r}	{p,q,r}
	{p,r}F	{p}	{p,q}
	{q,r}F	{r}	{r}
	{p,q,r}F	{p,r}	{p,q,r}

0. Copy the transition table

(for reference)

1. add Row(S) to table,

/* where S is the set of start states */

2. ToDO = {X|X in Row(S).tail } – {S}

3. While (ToDO != {}) {

3.1 S1 = ToDo.pop() ; // remove any element from D.

3.2 add(Row(S1)) to table

3.3 for(T in Row(S1).tail) {

if(!table.contains(T))

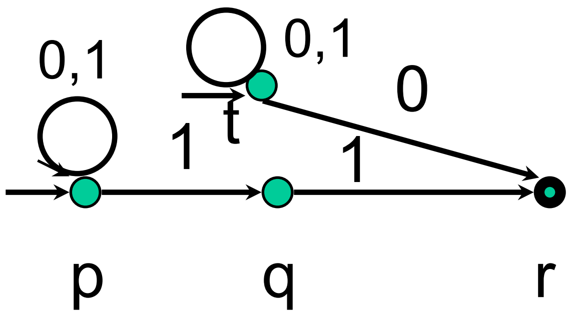
D = D U {T} ;

}

- Suppose p is a state and T is a set of states, then
  Row(p) = $(\{p\}, \delta(p,0),\delta(p,1))$
  Row(T) = $\cup_{p\in T}$ row(p) = $(T, \cup_{p\in T}\delta(p,0),\cup_{\pi\in T}\delta(p,1))$
- if r = (A,B,C) is a row, then r.tail = {B, C}

More example (step 0)

NFA:



		0	1
0.	→ p	{p}	{p,q}
	→ t	{t,r}	{t}
	q	{}	{r}
	rF	{}	{}

0. Copy the transition table (for reference)

1. add Row(S) to table,

/* where S is the set of start states */

2. ToDO = {X|X in Row(S).tail } – {S}

3. While (ToDO != {}) {

3.1 S1 = ToDo.pop() ; // remove any element from D.

3.2 add(Row(S1)) to table

3.2 for(T in Row(S1).tail) {

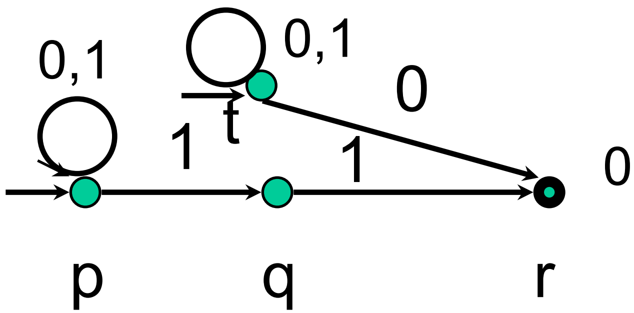
if(!table.contains(T))

D = D U {T} ;

}

More example (step 1)

NFA:



	0	1
→ p	{p}	{p,q}
→ t	{t,r}	{t}
q	{}	{r}
rF	{}	{}
1. {p,t}	{p,t,r}	{p,t,q}

- row(S) = row({p,t}) = row(p) U row(t)

0. **Copy the transition table (for reference)**

1. add Row(S) to table,

/* where S is the set of start states */

2. ToDO = {X|X in Row(S).tail } – {S}

3. While (ToDO != {}) {

3.1 S1 = ToDo.pop() ; // remove any element from D.

3.2 add(Row(S1)) to table

3.2 for(T in Row(S1).tail) {

if(!table.contains(T))

D = D U {T} ;

}

- Suppose p is a state and T is a set of states, then

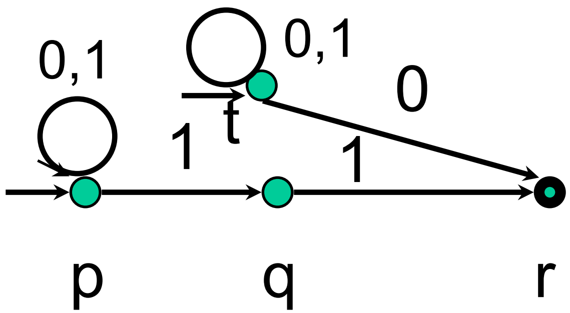
Row(p) = $(\{p\}, \delta(p,0), \delta(p,1))$

Row(T) = $\cup_{p\in T}$ row(p) $= (T, \cup_{p\in T}\delta(p,0), \cup_{\pi\in T}\delta(p,1))$

- if r = (A,B,C) is a row, then r.tail = {B, C}

More example (step 2)

NFA:



		0	1
0.	→ p	{p}	{p,q}
	→ t	{t,r}	{t}
	q	{}	{r}
	rF	{}	{}
1.	{p,t}	{p,t,r}	{p,t,q}
2.	{p,t,r}		
	{p,t,q}		

ToDO

0. Copy the transition table (for reference)

1. add Row(S) to table,

/* where S is the set of start states */

2. ToDO = {X|X in Row(S).tail } – {S}

3. While (ToDO != {}) {

   3.1 S1 = ToDo.pop() ; // remove any element from D.

   3.2 add(Row(S1)) to table

   3.2 for(T in Row(S1).tail) {

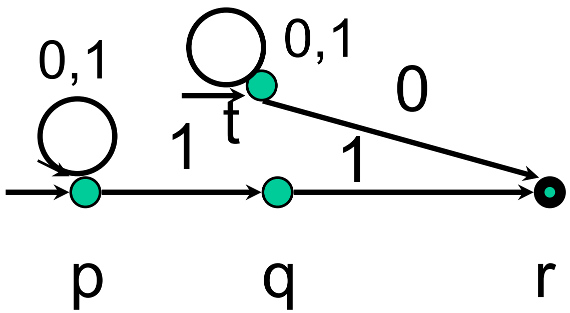
   if(!table.contains(T))

   D = D U {T} ;

}

More example (step 3.1,3.2)

NFA:



		0	1
0.	→ p	{p}	{p,q}
	→ t	{t,r}	{t}
	q	{}	{r}
	rF	{}	{}
1.	{p,t}	{p,t,r}	{p,t,q}
3.	**{p,t,r}**	{p,t,r}	{p,t,q}
	{p,t,q}		

- row({p,t,r}) = row(p) U row(t) U row(r)

0. Copy the transition table (for reference)

1. add Row(S) to table,

/* where S is the set of start states */

2. ToDO = {X|X in Row(S).tail } – {S}

3. While (ToDO != {}) {

3.1 S1 = ToDo.pop() ; // remove any element from D.

3.2 add(Row(S1)) to table

3.3 for(T in Row(S1).tail) {

if(!table.contains(T))

D = D U {T} ;

}

- Suppose p is a state and T is a set of states, then

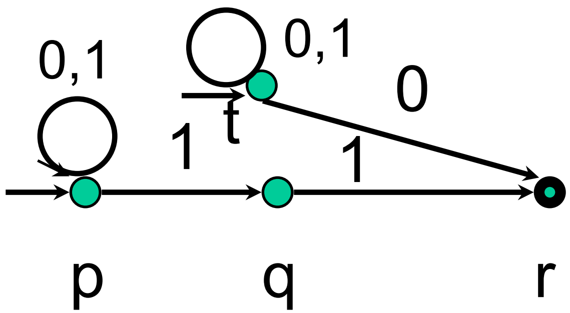
Row(p) = $(\{p\}, \delta(p,0), \delta(p,1))$

Row(T) = $\mathrm{U}_{p\in T}$ row(p) = $(T, \cup_{p\in T}\delta(p,0), \cup_{\pi\in T}\delta(p,1))$

- if r = (A,B,C) is a row, then r.tail = {B, C}

More example (step 3.3)

NFA:



		0	1
0.	→ p	{p}	{p,q}
	→ t	{t,r}	{t}
	q	{}	{r}
	rF	{}	{}
1.	{p,t}	{p,t,r}	{p,t,q}
3.	{p,t,r}	{p,t,r}	{p,t,q}
	{p,t,q}		

0. Copy the transition table (for reference)

1. add Row(S) to table,

/* where S is the set of start states */

2. ToDO = {X|X in Row(S).tail } – {S}

3. While (ToDO != {}) {

3.1 S1 = ToDo.pop() ; // remove any element from D.

3.2 add(Row(S1)) to table

3.3 for(T in Row(S1).tail) {

if(!table.contains(T))

D = D U {T} ;

}

- Suppose p is a state and T is a set of states, then

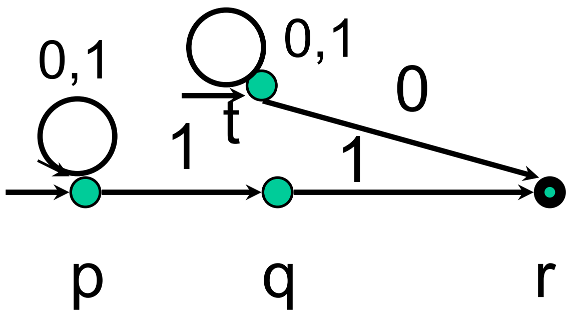
Row(p) = ({p}, δ(p,0),δ(p,1))

Row(T) = $U_{p\in T}$ row(p) = (T, $\cup_{p\in T}\delta(p,0),\cup_{\pi\in T}\delta(p,1)$)

- if r = (A,B,C) is a row, then r.tail = {B, C}

More example (step 3.1 &3.2)

NFA:



		0	1
0.	→ p	{p}	{p,q}
	→ t	{t,r}	{t}
	q	{}	{r}
	rF	{}	{}
1.	{p,t}	{p,t,r}	{p,t,q}
3.	{p,t,r}	{p,t,r}	{p,t,q}
	{p,t,q}	{p,t,r}	{p,t,q,r}

0. Copy the transition table (for reference)

1. add Row(S) to table,

/* where S is the set of start states */

2. ToDO = {X|X in Row(S).tail } – {S}

3. While (ToDO != {}) {

3.1 S1 = ToDo.pop() ; // remove any element from D.

3.2 add(Row(S1)) to table

3.3 for(T in Row(S1).tail) {

if(!table.contains(T))

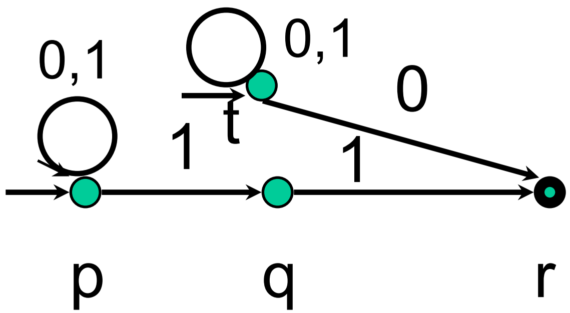
D = D U {T} ;

}

- Suppose p is a state and T is a set of states, then
  Row(p) = $(\{p\}, \delta(p,0), \delta(p,1))$
  Row(T) = $\cup_{p\in T}$ row(p) = $(T, \cup_{p\in T}\delta(p,0), \cup_{\pi\in T}\delta(p,1))$
- if r = (A,B,C) is a row, then r.tail = {B, C}

More example (step 3.3)

NFA:



		0	1
0.	→ p	{p}	{p,q}
	→ t	{t,r}	{t}
	q	{}	{r}
	rF	{}	{}
1.	{p,t}	{p,t,r}	{p,t,q}
3.	{p,t,r}	{p,t,r}	{p,t,q}
	{p,t,q}	{p,t,r}	{p,t,q,r}
	{p,t,q,r}		

ToDO

0. **Copy the transition table (for reference)**

1. **add Row(S) to table,**

/* where S is the set of start states */

2. **ToDO = {X|X in Row(S).tail } – {S}**

3. While (ToDO != {}) {

3.1 S1 = ToDo.pop() ; // remove any element from D.

3.2 add(Row(S1)) to table

3.3 for(T in Row(S1).tail) {

if(!table.contains(T))

D = D U {T} ;

}

- Suppose p is a state and T is a set of states, then

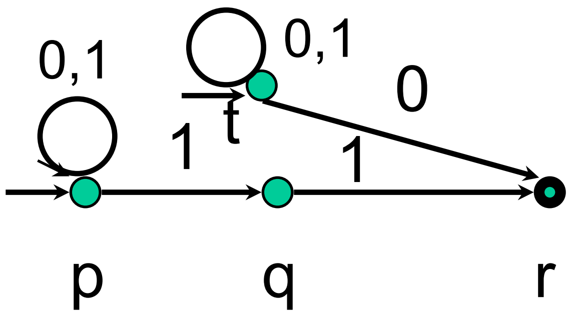
Row(p) = $(\{p\}, \delta(p,0), \delta(p,1))$

Row(T) = $U_{p\in T}$ row(p) = $(T, \cup_{p\in T}\delta(p,0), \cup_{\pi\in T}\delta(p,1))$

- if r = (A,B,C) is a row, then r.tail = {B, C}

More example (step 3.1&2)

NFA:



		0	1
0.	→ p	{p}	{p,q}
	→ t	{t,r}	{t}
	q	{}	{r}
	rF	{}	{}
1.	{p,t}	{p,t,r}	{p,t,q}
3.	{p,t,r}	{p,t,r}	{p,t,q}
	{p,t,q}	{p,t,r}	{p,t,q,r}
	{p,t,q,r}	{p,t,r}	{p,t,q,r}

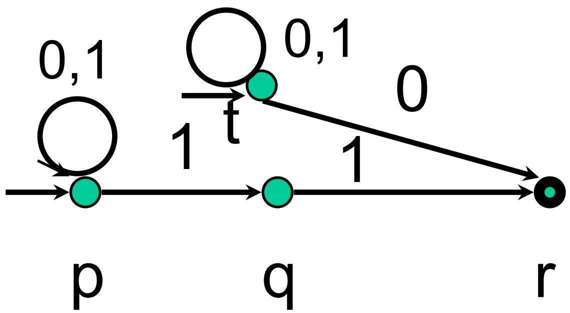
0. Copy the transition table (for reference)
1. add Row(S) to table, /* where S is the set of start states */
2. ToDO = {X|X in Row(S).tail } – {S}
3. **While (ToDO != {}) {**
   - **3.1 S1 = ToDo.pop() ; // remove any element from D.**
   - **3.2 add(Row(S1)) to table**
   - **3.3 for(T in Row(S1).tail) {**
     - **if(!table.contains(T))**
       - **D = D U {T} ;**

}

- Suppose p is a state and T is a set of states, then
  Row(p) = ({p}, δ(p,0),δ(p,1))
  Row(T) = $U_{p\in T}$ row(p) = (T, $\cup_{p\in T}\delta(p,0),\cup_{\pi\in T}\delta(p,1)$)
- if r = (A,B,C) is a row, then r.tail = {B, C}

More example (step 3.3)

NFA:



		0	1
0.	→ p	{p}	{p,q}
	→ t	{t,r}	{t}
	q	{}	{r}
	rF	{}	{}
1.	{p,t}	{p,t,r}	{p,t,q}
3.	{p,t,r}	{p,t,r}	{p,t,q}
	{p,t,q}	{p,t,r}	{p,t,q,r}
	{p,t,q,r}	{p,t,r}	{p,t,q,r}

0. Copy the transition table (for reference)

1. add Row(S) to table,
 /* where S is the set of start states */

2. ToDO = {X|X in Row(S).tail } – {S}

3. While (ToDO != {}) {

3.1 S1 = ToDo.pop() ; // remove any element from D.

3.2 add(Row(S1)) to table

3.3 for(T in Row(S1).tail) {

if(!table.contains(T))

D = D U {T} ;

}

- Suppose p is a state and T is a set of states, then
Row(p) = $(\{p\}, \delta(p,0), \delta(p,1))$
Row(T) = $U_{p\in T}$ row(p) = $(T, \cup_{p\in T}\delta(p,0), \cup_{\pi\in T}\delta(p,1))$
- if r = (A,B,C) is a row, then r.tail = {B, C}

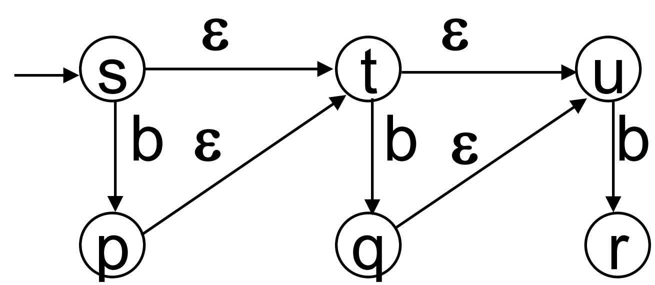
ε-transition

- Another extension of FAs, useful but adds no more power.
- An ε-transition is a transition with label ε, a label standing for the empty string ε.
- The FA can take such a transition anytime w/o reading an input symbol.

$$p \xrightarrow{\varepsilon} q$$

Ex 6.5 : The set accepted by the FA is {b,bb,bbb}.

Ex 6.6 : A NFA-ε accepting the set $\{x \in \{a\}^* \mid |x| \text{ is dividable by 3 or 5}\}$.



Ex6.5

- real advantage of ε -transition:
  - convenient for specification
  - add no extra power

NFA-ε

- N = (Q,Σ,δ,S,F) : a NFA-ε,where
  - Q, Σ, S and F are the same as NFA,
  - $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$.
- The set Eclosure(A) is the set of ref. and transitive closure of the ε-transition of A =

  $\{ q \in Q \mid \exists\ \varepsilon\text{-path } p - p_1 - p_2 \ldots - p_n \text{ with } p \in A \text{ and } p_n = q \}$

Note: Eclosure(A) (abbreviated as EC(A)) = EC(EC(A)).

- The multistep version of δ is modified as follows:
  - $\Delta: 2^Q \times \Sigma^* \to 2^Q$ where, for all $A \subseteq Q$, $y \in \Sigma^*$, $a \in A$
  - $\Delta(A, \varepsilon) = \text{Eclosure}(A)$
  - $\Delta(A, ya) = \bigcup_{p \in \Delta(A,y)} \text{Eclosure}(\delta(p,a))$
- $L(N) = \{ x \mid \Delta(S, x) \cap F \neq \{\} \}$ //The language accepted by N

E-closure

- **Eclosure(A) is the set of states reachable from states of A without consuming any input symbols,**

(i.e., $q \in$ Eclosure(A) iff $\exists p \in A$ s.t. $q \in \Delta(p, \varepsilon^k)$ for some $k \geq 0$).

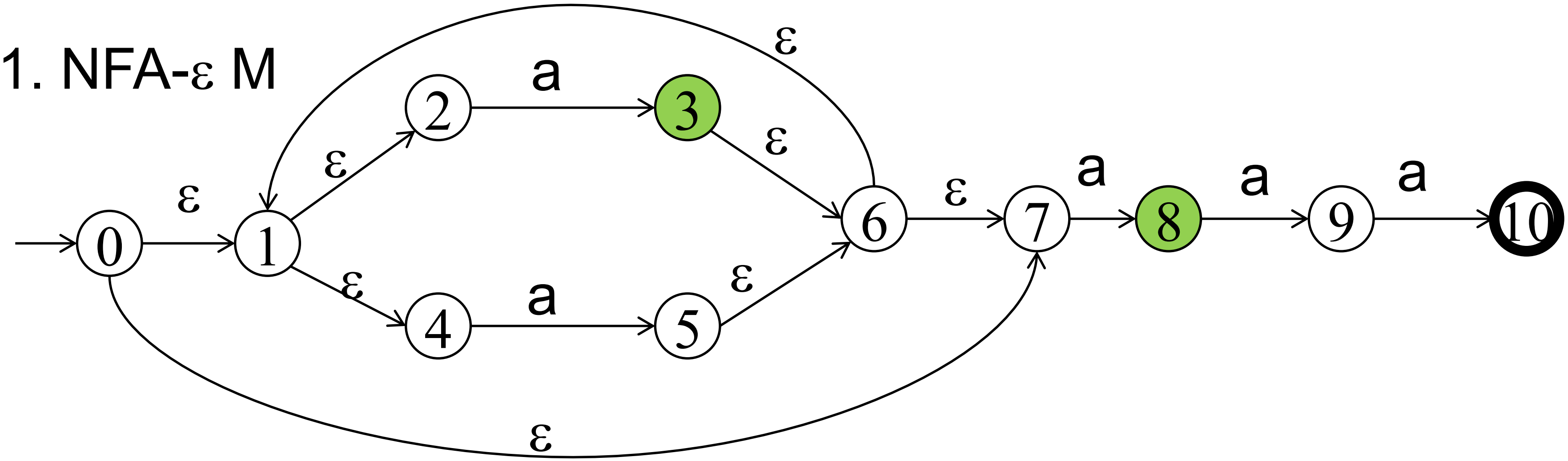
- **Eclosure(A) can be computed as follows:**

```
Result = A;
add all states in A to a queue //or stack!
while(! queue.empty()) {
   s = queue.remove();
   for each q ∈ δ(s,ε) do
      if(q∉Result){ Result = Result U {q} ;
                    queue.add(q)              }
return Result
```

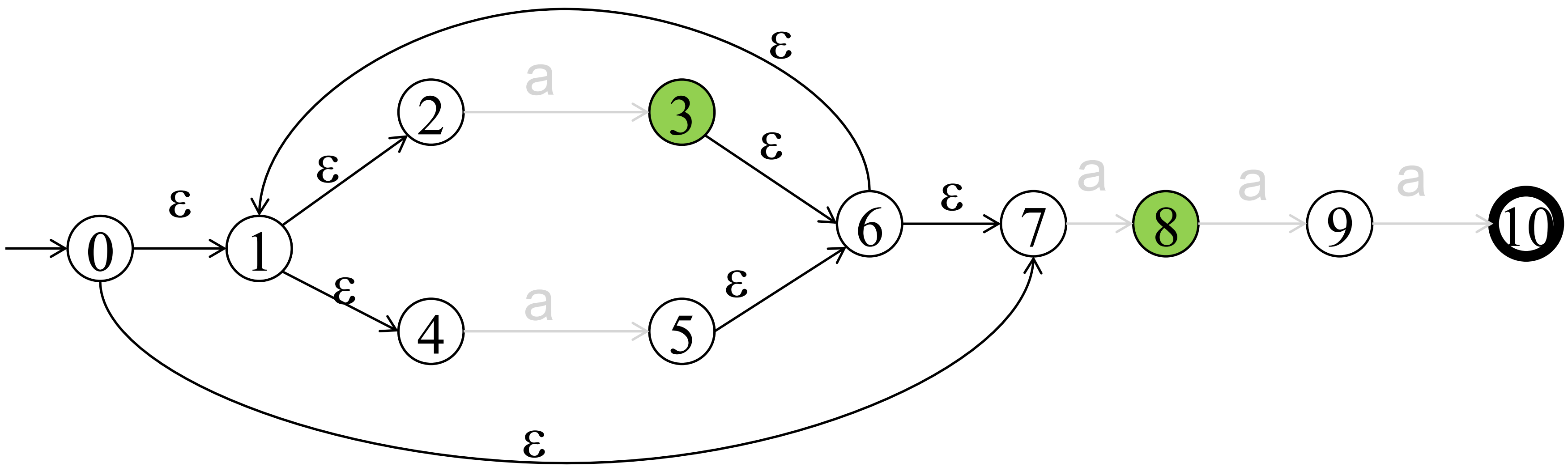
Note: We can precompute the reflexive&transitive closure matrix T* of the ε-transition matrix T of the NFA, and use the result to get Eclosure(A) = $\{q \mid \exists p \in A$ s.t. $T^*(p,q) = 1\}$ for every required set A.

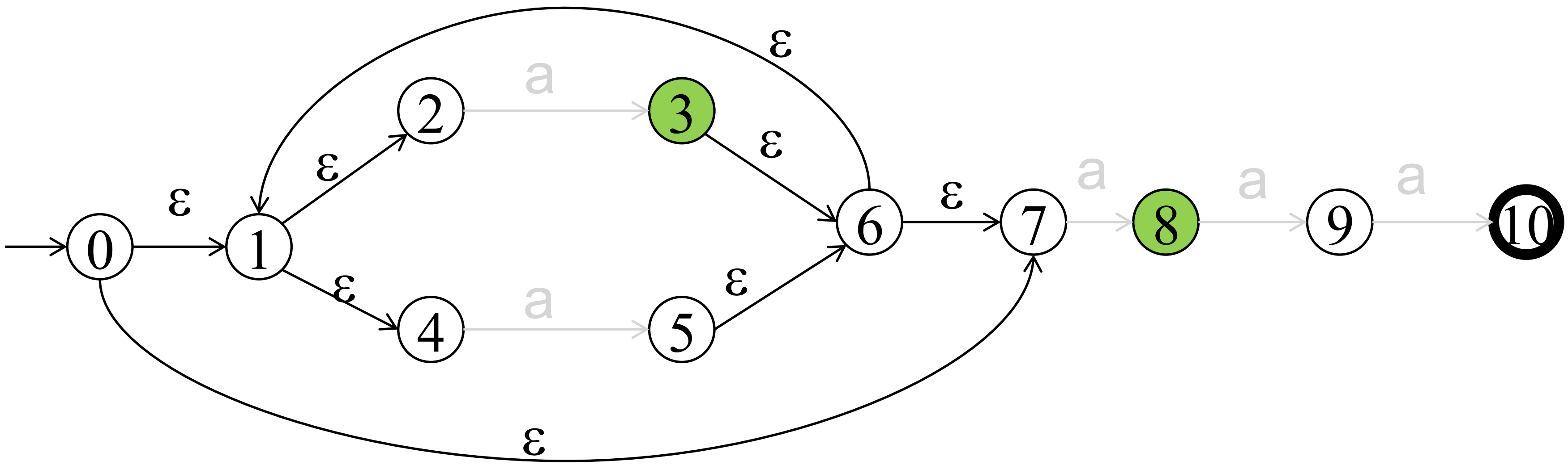
Example : compute Eclosure

1. NFA-ε M

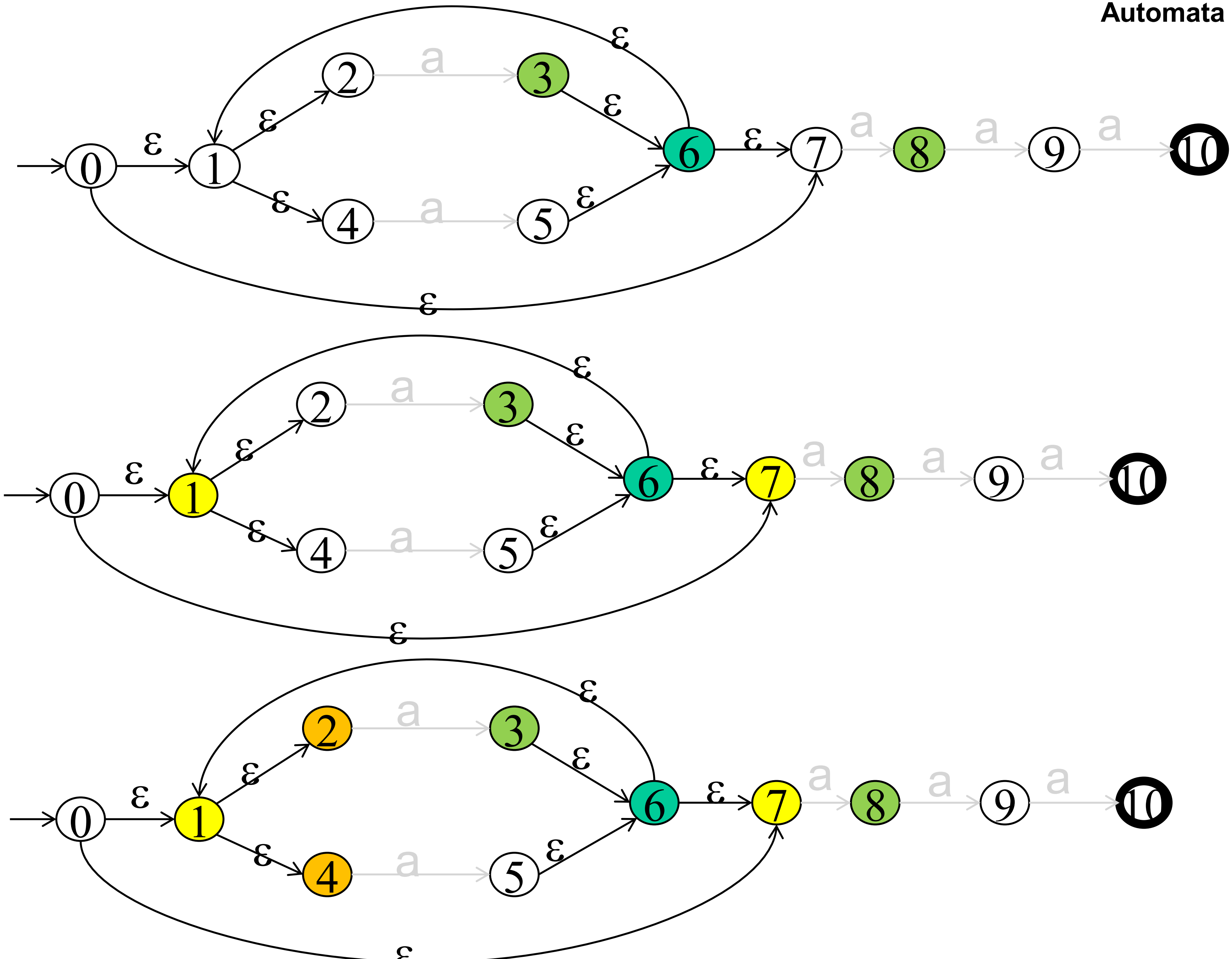


2. NFA M with non ε–transitions removed





- EC({3,8}) is computed as follows:
- Intially (Result,Queue) = ({3,8}, [3,8])
- 3→6 => ({3,8,6}, [8,6]) 8 →{} =>({3,8,6}, [6])
- 6→1,7 => ({3,8,6,1},[1]) => ({3,8,6,1,7}, [1,7])
- 1→2,4 =>({3,8,6,1,7,2}, [7,2]) =>({3,8,6,1,7,2,4},[7,2,4])
- 7→{},2→{},4→{} => ({3,8,6,1,7,2,4}, [])


0
ε
1
ε
2
a
3
ε
ε
6
ε
7
a
8
a
9
a
10
ε
4
a
5
ε
ε

The subset construction for NFA-ε

- $N = (Q_N,\Sigma,\delta_N,S_N,F_N)$: a NFA-ε, where $\delta_N : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$.
- $M = (Q_M,\Sigma,\delta_M,s_M,F_M)$ (denoted 2^N): a DFA where
  - $Q_M = \{ EC(A) \mid A \subseteq Q_N \}$
  - $\delta_M(A,a) = \bigcup_{q\in A} EC(\delta_N(q,a))$ for every $A \in Q_M$.
  - $s_M = EC(S_N)$ and
  - $F_M = \{A \in Q_M \mid A \cap F_N \neq \{\}\}$.
  - note: States of M are subsets of states of N.
- Lem 6.3: For any $A \subseteq Q_N$ and $x \in \Sigma^*$, $\Delta_M(A,x) = \Delta_N(A,x)$.

pf: by ind on $|x|$. if $x = \varepsilon \Rightarrow \Delta_M(A,\varepsilon) = A = EC(A) = \Delta_N(A,\varepsilon)$. --(def)

if $x = ya \Rightarrow \Delta_M(A,ya) = \delta_M(\Delta_M(A,y),a)$ -- (def)

$= \delta_M(\Delta_N(A,y),a)$ -- ind. hyp.

$= \bigcup_{q \in \Delta N(A,y)} EC(\delta_N(q,a))$ -- def of δ_M

$= \Delta_N(A, ya)$ – def of Δ_N

Theorem 6.4: M and N accept the same set.

pf: $x \in L(M)$ iff $\Delta_M(s_M,x)\in F_M$ iff $\Delta_N(EC(S_N),x)\cap F_N \neq \{\}$ iff $x \in L(N)$.

More closure properties

- If A and B are regular languages, then so are AB and A*.
- $M = (Q_1,\Sigma,\delta_1,S_1,F_1)$, $N=(Q_2,\Sigma,\delta_2,S_2,F_2)$: two NFAs
- The machine $M \bullet N$, which firstly executes M and then executes N, can be defined as follows:
- $M \bullet N =_{def} (Q, \Sigma, \delta, S, F)$ where
  - Q = disjoint union of Q_1 and Q_2,
  - $S = S_1$,
  - $F = F_2$,
  - $\delta = \delta_1 \cup \delta_2 \cup \{ (p, \varepsilon, q) \mid p \in F_1 \text{ and } q \in S_2 \}$
- Lemma:
  - 1. $x \in L(M)$ and $y \in L(N)$ imply $xy \in L(MN)$
  - 2. $x \in L(MN) \Rightarrow \exists\, y,z$ s.t. $x = yz$ and $y \in L(M)$ and $z \in L(N)$.
- Corollary: $L(M \bullet N) = L(M) \bullet L(N)$

M* machine

- $M = (Q_1,\Sigma,\delta_1,S_1,F_1)$: a NFA
- The machine M*, which executes M a nondeterministic number of times, can be defined as follows:
- $M^* =_{def} (Q, \Sigma, \delta, S, F)$ where
  - $Q = Q \cup \{s,f\}$, where s and f are two new states $\notin Q$
  - $S = \{s\}$, $F = \{f\}$,
  - $\delta = \delta_1 \cup \{(s, \varepsilon, f)\} \cup \{(s,\varepsilon,p) \mid p \in S_1\} \cup \{(q,\varepsilon,s) \mid q \in F_1\}$

Theorem: $L(M^*) = L(M)^*$