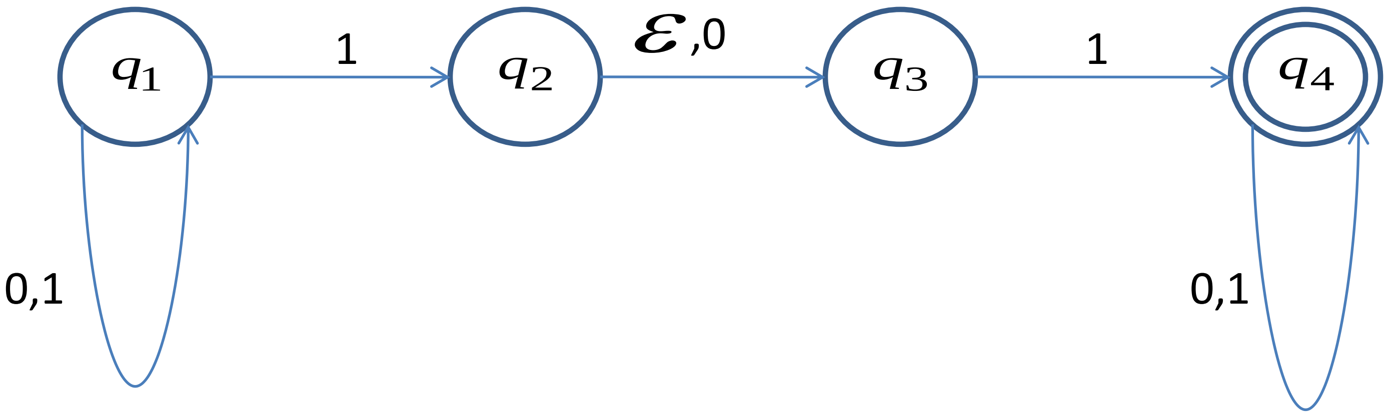
Non-Deterministic Finite Automata

N Geetha
AM & CS
PSG Tech

Example of an NFA

NFA – Nondeterministic Finite Automaton



1. A state nay have 0 or more transitions labeled with the same symbol.
2. ε Transitions are possible.
3. Can have multiple initial states

Computation of an NFA

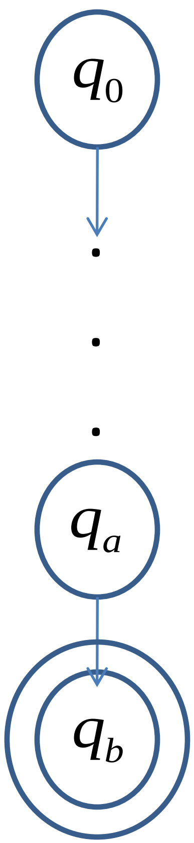
- When several transitions with the same label exist, an input word may induce **several** paths.
- When no transition is possible a computation is “stuck”.

Q: Which words are accepted and which are not?

A: If word w induces (at least) a single accepting path, the automaton “chooses” this **accepting path** and w is accepted.

Possible Computations

DFA

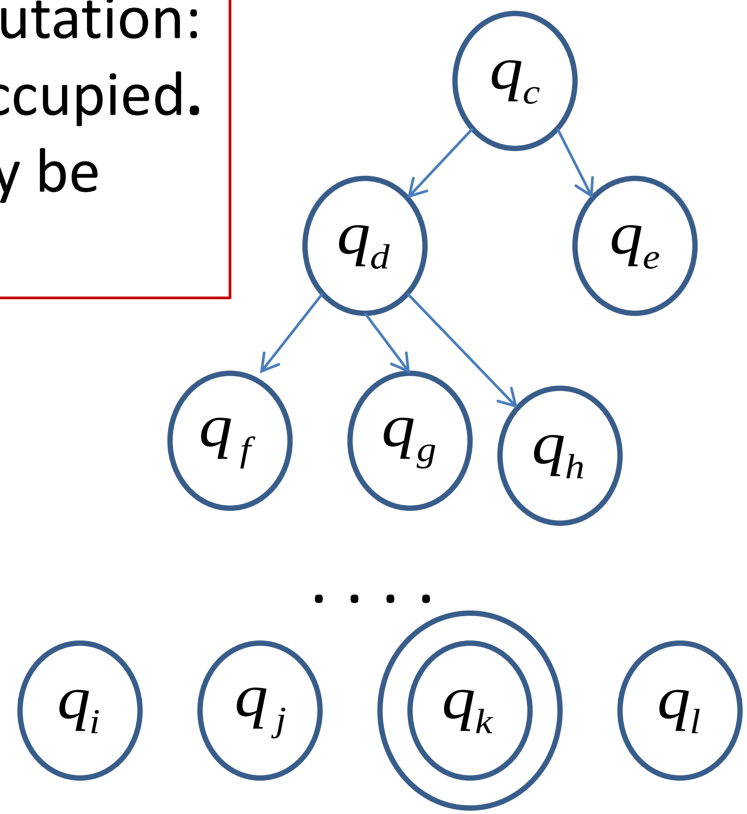


At each step of the computation:

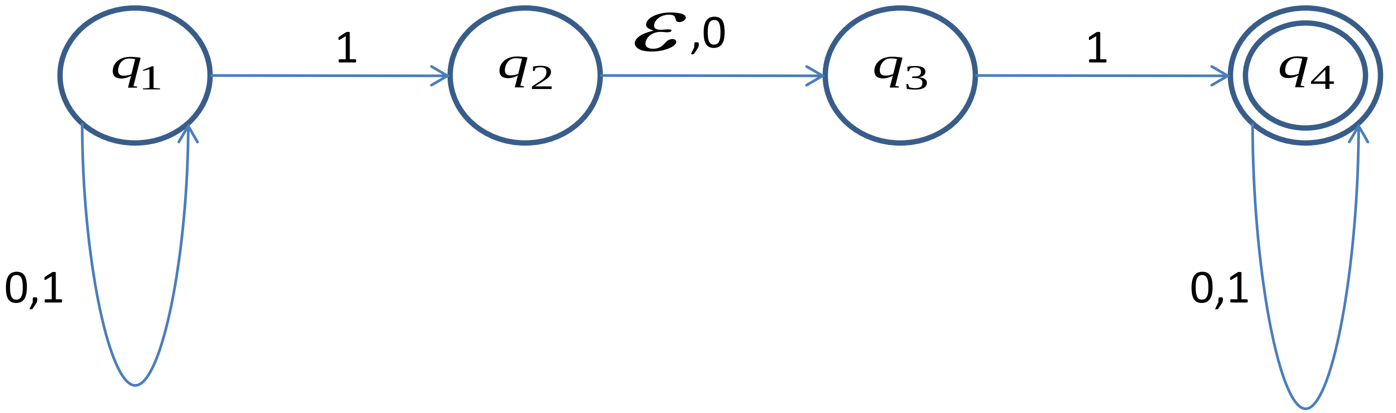
DFA - A **single state** is occupied.

NFA - **Several states** may be occupied.

NFA



Computation of an NFA - Example

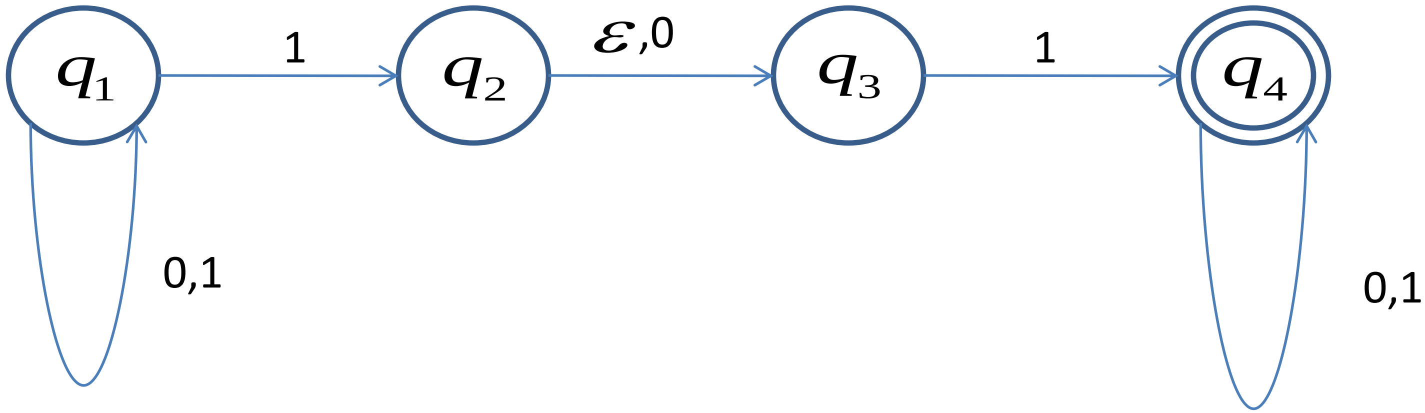


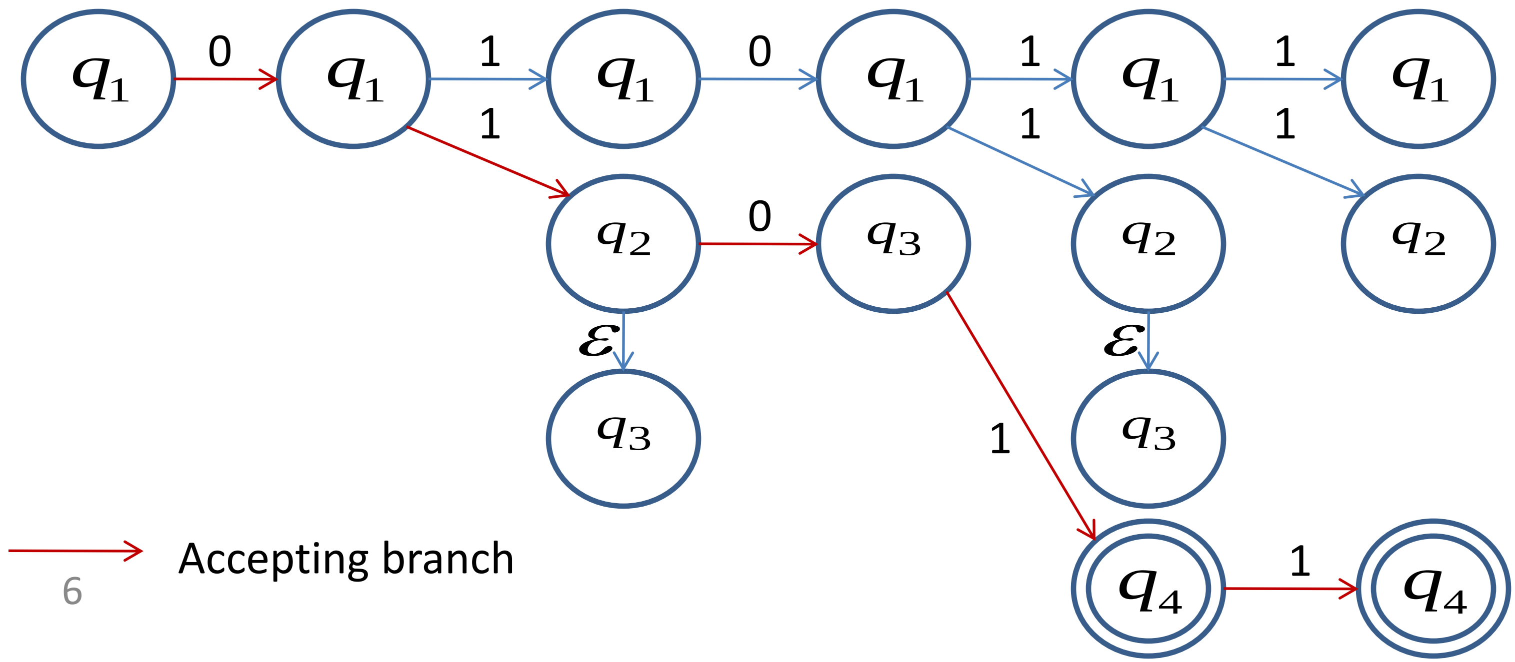
On the input word w=01011 there exist an accepting path and w is accepted.

Can we characterize (find) the language recognized by this automaton?

Words containing either 101 or 11 as substrings.

Computation tree of 01011





Accepting branch

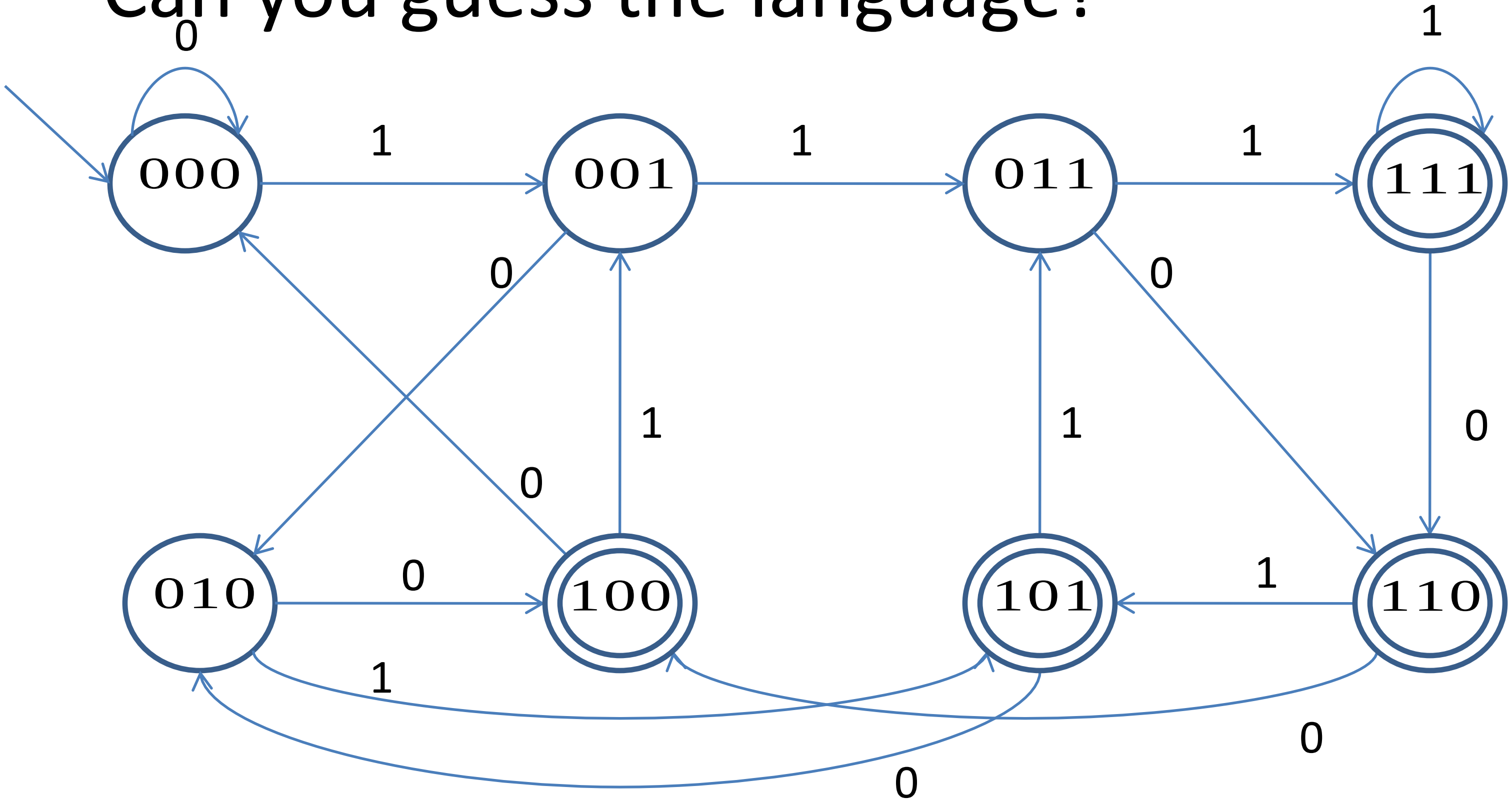
Why do we Care About NFAs?

- NFA-s and DFA-s are **equivalent** (Meaning: They recognize the same set of languages). In other words: Each NFA recognizing language L has an equivalent DFA recognizing L.

  (**Note:** This statement **must be proved**.)
- But usually, the NFA is much simpler.
- Enables the proof of theorems. (e.g. about regular operations)

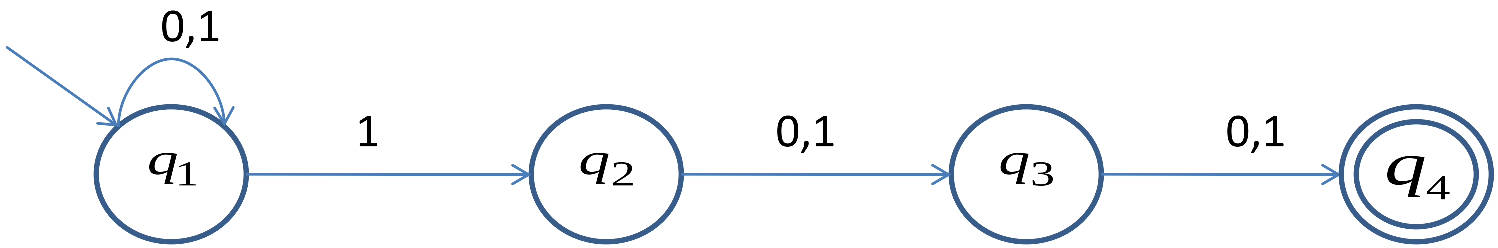
Example - A Complicated DFA

- Can you guess the language?



Example – An Equivalent NFA

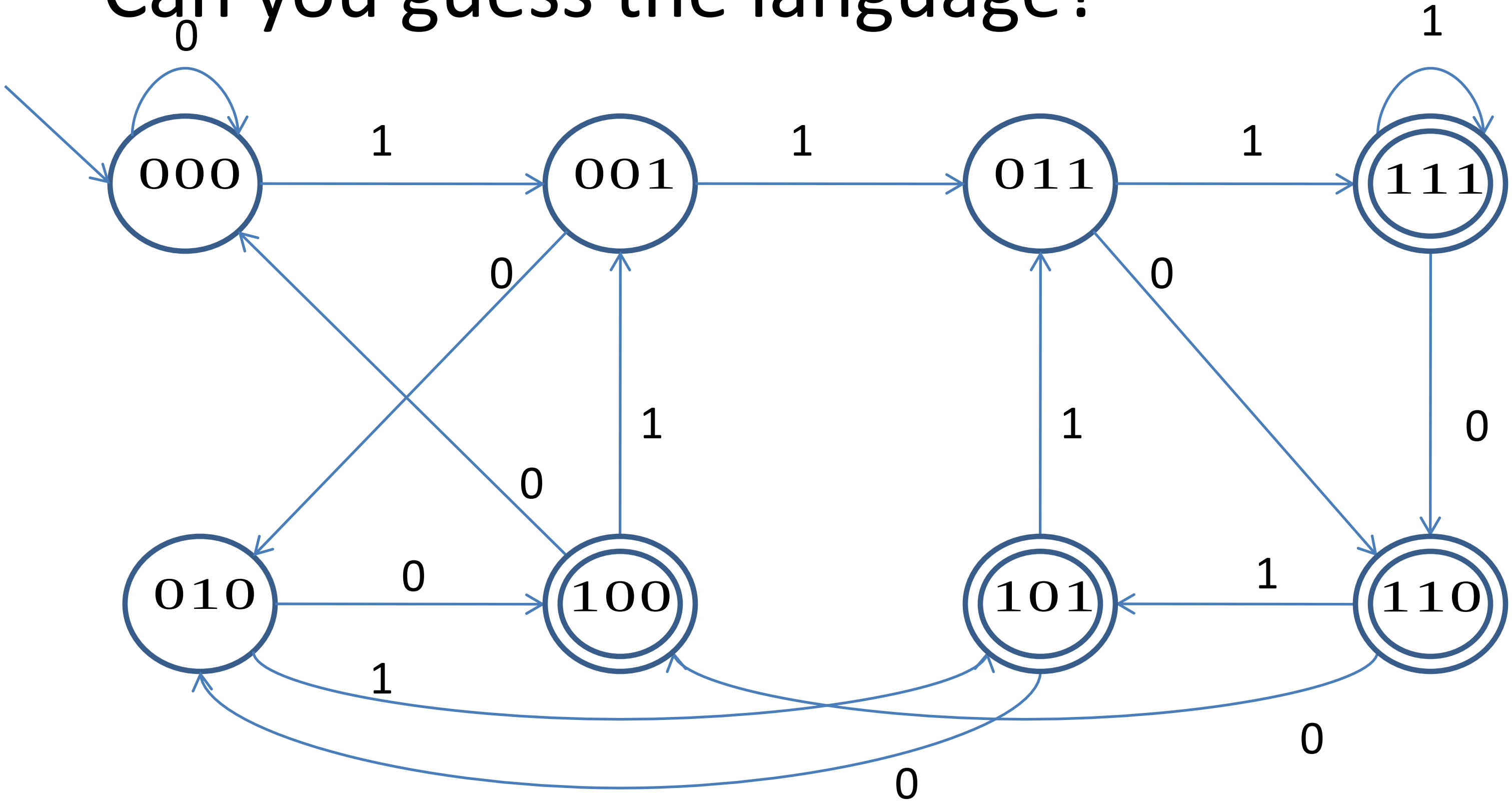
Can you guess the language?



Bit Strings that have 1 in position third from the end

Example - A Complicated DFA

- Can you guess the language?



Can you verify it now?

DFA – A Formal Definition(Rerun)

A ***finite automaton*** is a 5-tupple $(Q, \Sigma, \delta, q_0, F)$ where:

1. Q is a finite set called the ***states***.
2. Σ is a finite set called the ***alphabet***.
3. $\delta : Q \times \Sigma \to Q$ is the ***transition function***.
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the set of ***accepting states***.

NFA – A Formal Definition

A ***finite automaton*** is a 5-tupple $(Q, \Sigma, \delta, q_0, F)$ where:

1. Q is a finite set called the ***states***.
2. Σ is a finite set called the ***alphabet***.
3. $\boxed{\delta : Q \times \Sigma_\varepsilon \to P(Q) \text{ is the } \textbf{\textit{transition function}}.}$ $\left(\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}\right)$
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the set of ***accept states***

Differences between NFA-s and DFA-s

There are **two** differences:

1. The range of the **transition function** δ is now $P(Q)$. (The set of **subsets** of the state set Q)
2. The transition function allows ε transitions.

Computations of NFA-s

In general a computation of an NFA, N, on input w, induces a **computation tree**.

Each path of the computation tree represents a possible computation of N.

The NFA N accepts w, if its computation tree includes **at least** one path ending with an accepting state.

Computations of NFA-s

There are two ways to look at computations of an NFA:

- The first is to say that the NFA N "**chooses**" the right path on its tree of possible computations.
- The second is to say that the NFA N traverses its computation tree **"in parallel"**.

Equivalence Between DFAs and NFAs

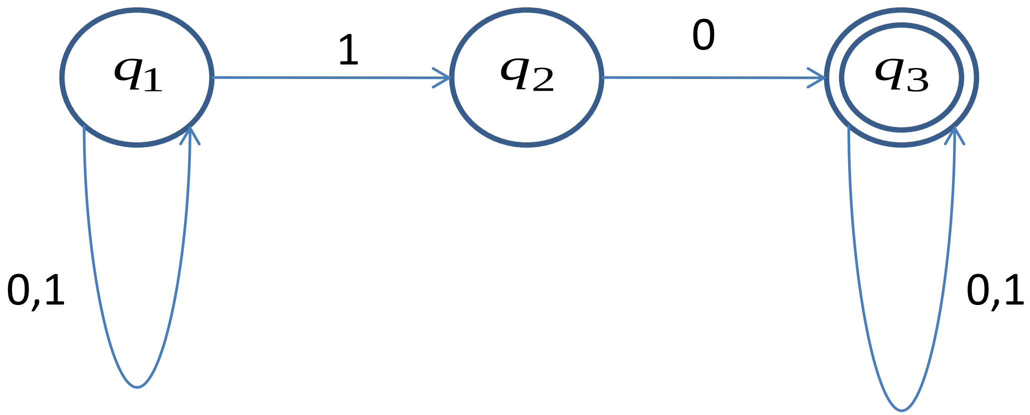
Now we prove that the class of NFAs is ***Equivalent*** to the class of DFA:

Theorem: For every NFA N, there exists a DFA $M = M(N)$, such that $L(N) = L(M(N))$.

Proof Idea: The proof is ***Constructive***: We assume that we know N, and construct a simulating DFA, M .

Construction Demonstration

We start with an NFA with no ε transitions.



The equivalent DFA is denoted by:

$$D = < Q', \Sigma, \delta', q_0', F' >$$

Construction Demonstration

The state set of the equivalent DFA, D, should reflect the fact that at each step of the computation, N may occupy **several sates**, Thus we define the state set of D as the **power-set** of the state set of N, namely:

$$Q' = P(Q) = \left\{ \begin{matrix} \phi & \{q_1\} & \{q_2\} & \{q_3\} \\ \{q_1, q_2\} & \{q_2, q_3\} & \{q_1, q_3\} & \{q_1, q_2, q_3\} \end{matrix} \right\}$$

Construction Demonstration

Recall that the transition function of any NFA is defined by $\delta : Q \times \Sigma_{\varepsilon} \to P(Q)$.

Do not be confused by the use of $P(Q)$ as the state set of D . D is a DFA, not the original NFA.

Construction Demonstration

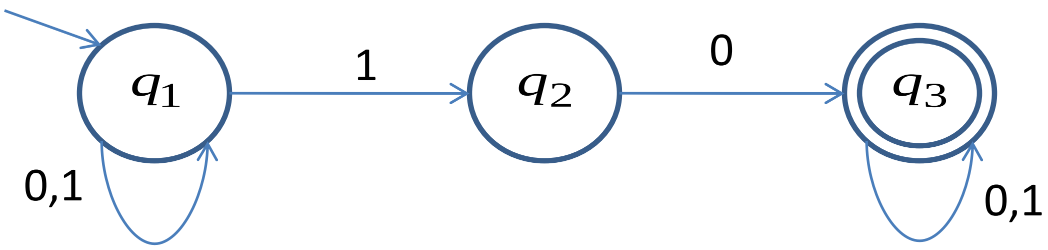
The alphabet of D, Σ , is identical to the alphabet of N.

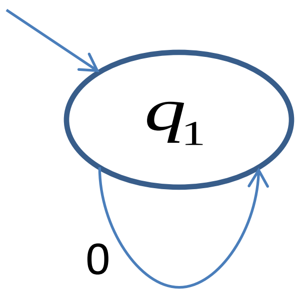
The starting state of D corresponds to the starting state of N, namely $\{q_0\}$.

The most complex part of the construction is the transition function:

In the sequel it is demonstrated stage by stage:

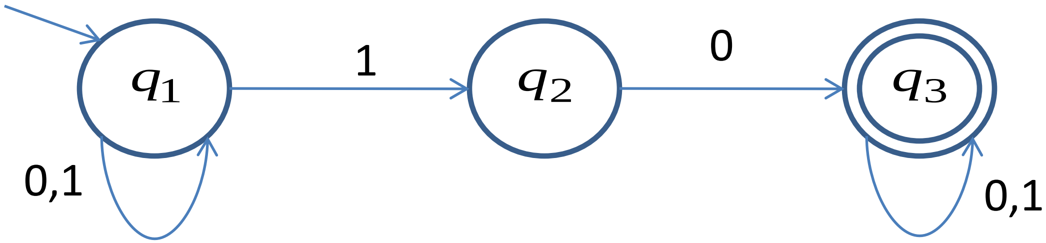
Construction Demonstration

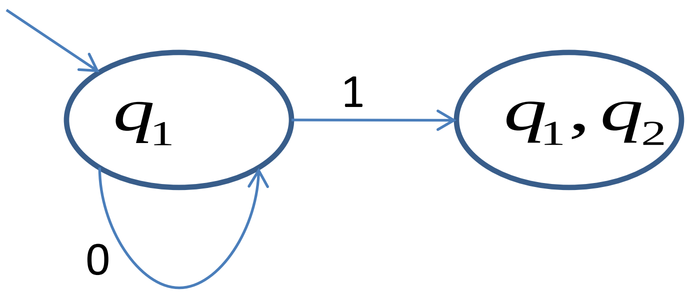




Since the only state reachable from q_1 by 0 is q_1, we get $\delta'(\{q_1\},0)=\{q_1\}$.

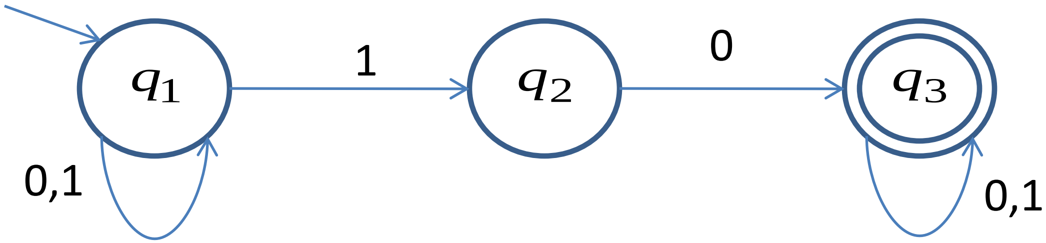
Construction Demonstration

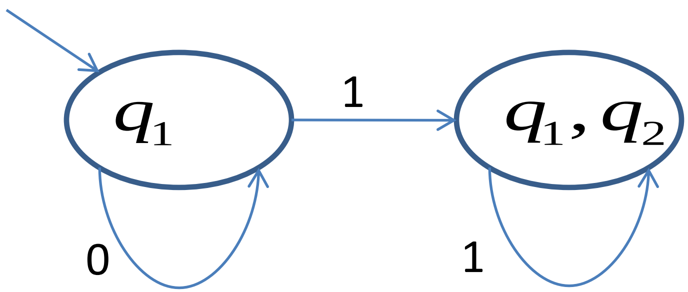




Since both q_1 and q_2 are reachable from q_1 by 1, we get $\delta'(\{q_1\},1) = \{q_1, q_2\}$.

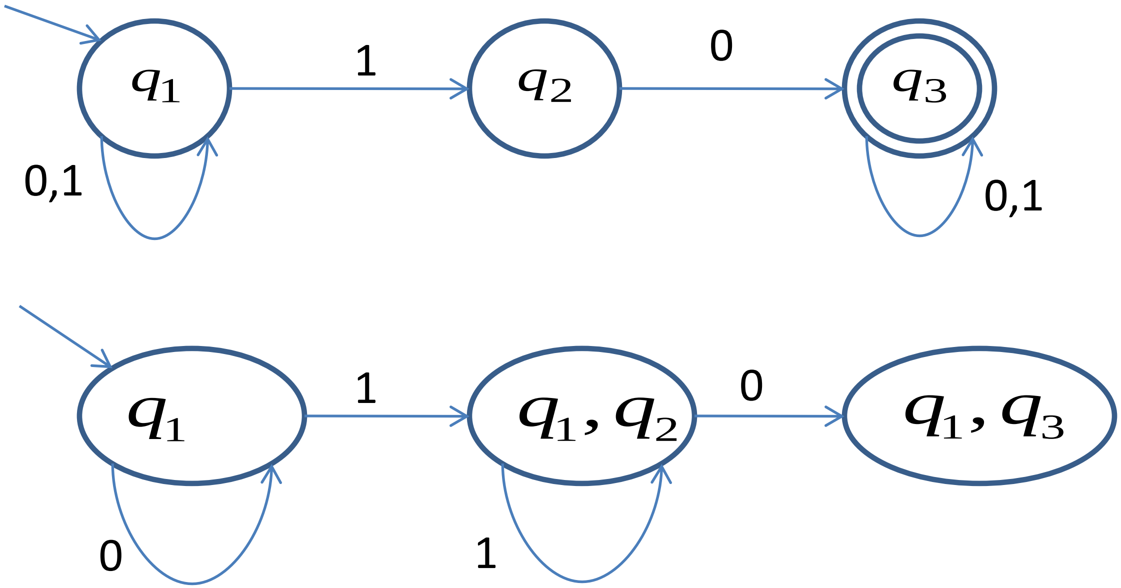
Construction Demonstration





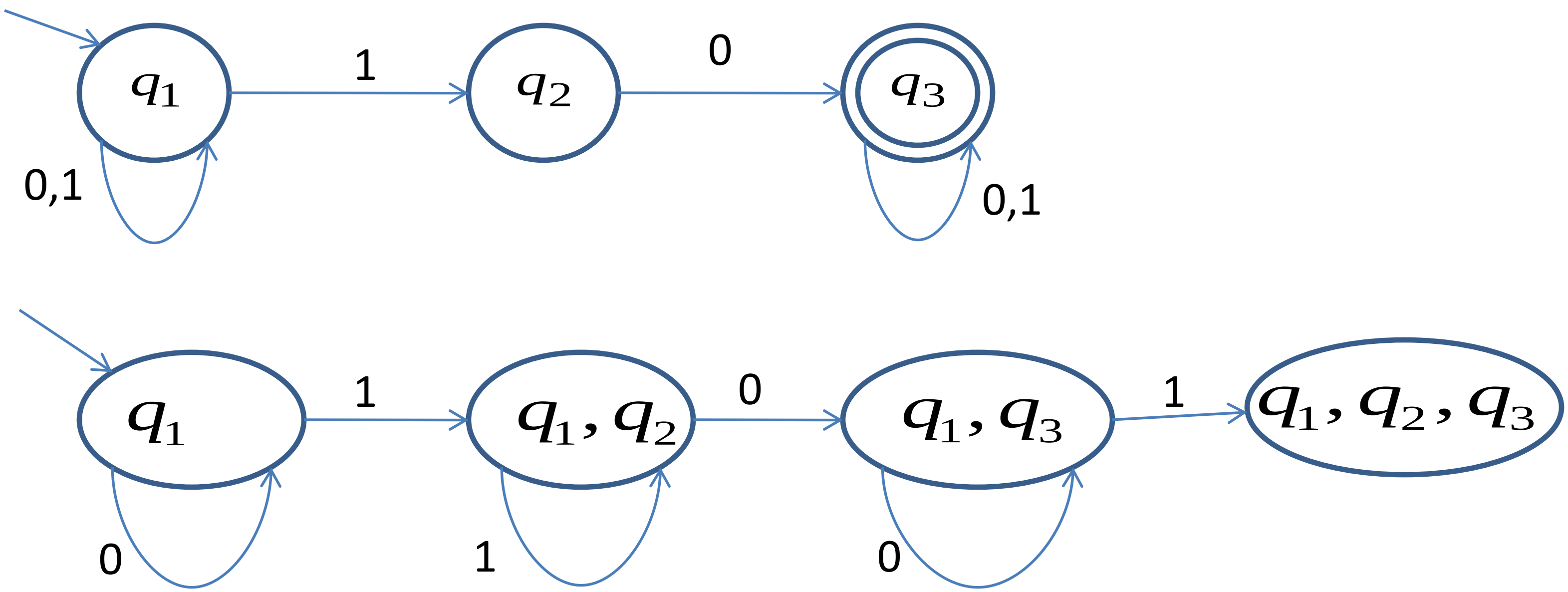
Similarly we get $\delta'(\{q_1, q_2\}, 1) = \{q_1, q_2\}$.

Construction Demonstration



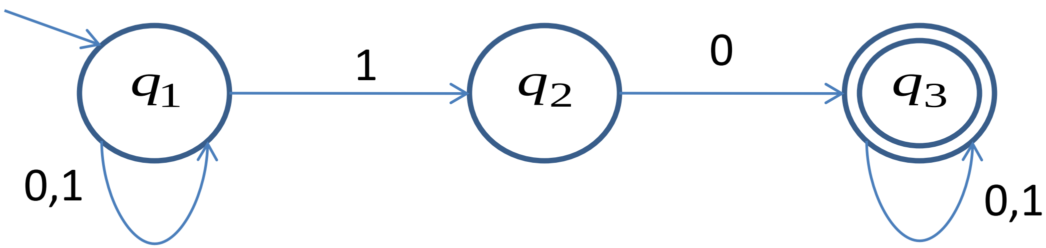
and $\delta'(\{q_1, q_2\}, 0) = \{q_1, q_3\}$.

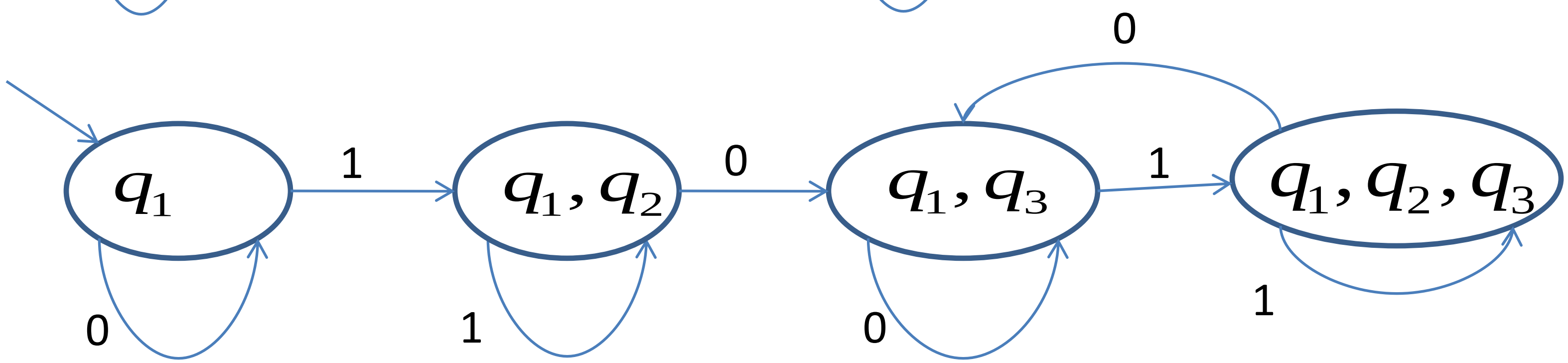
Construction Demonstration



Since $\delta(q_1, 1) = \{q_1, q_2\}$ and $\delta(q_3, 1) = \{q_3\}$,
we get $\delta'(\{q_1, q_3\}, 1) = \{q_1, q_2, q_3\}$.

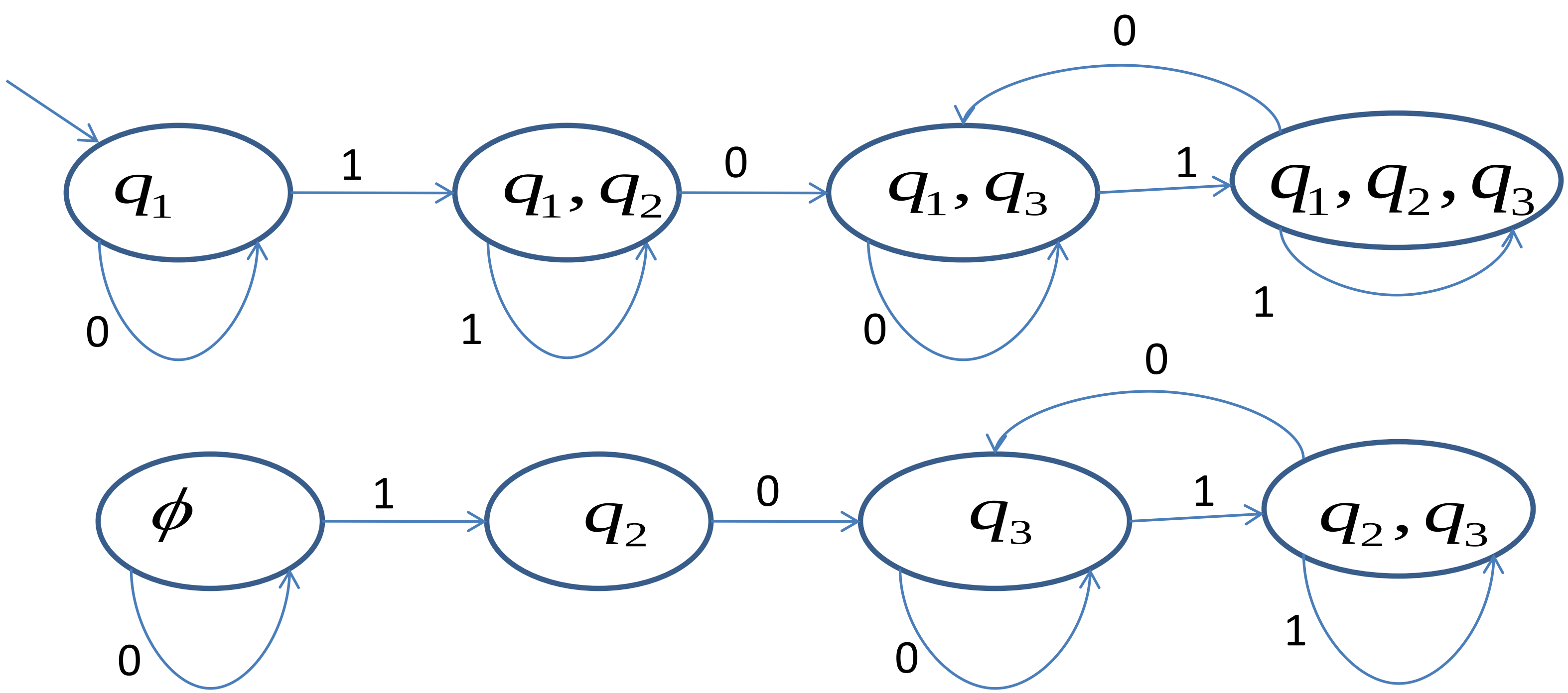
Construction Demonstration





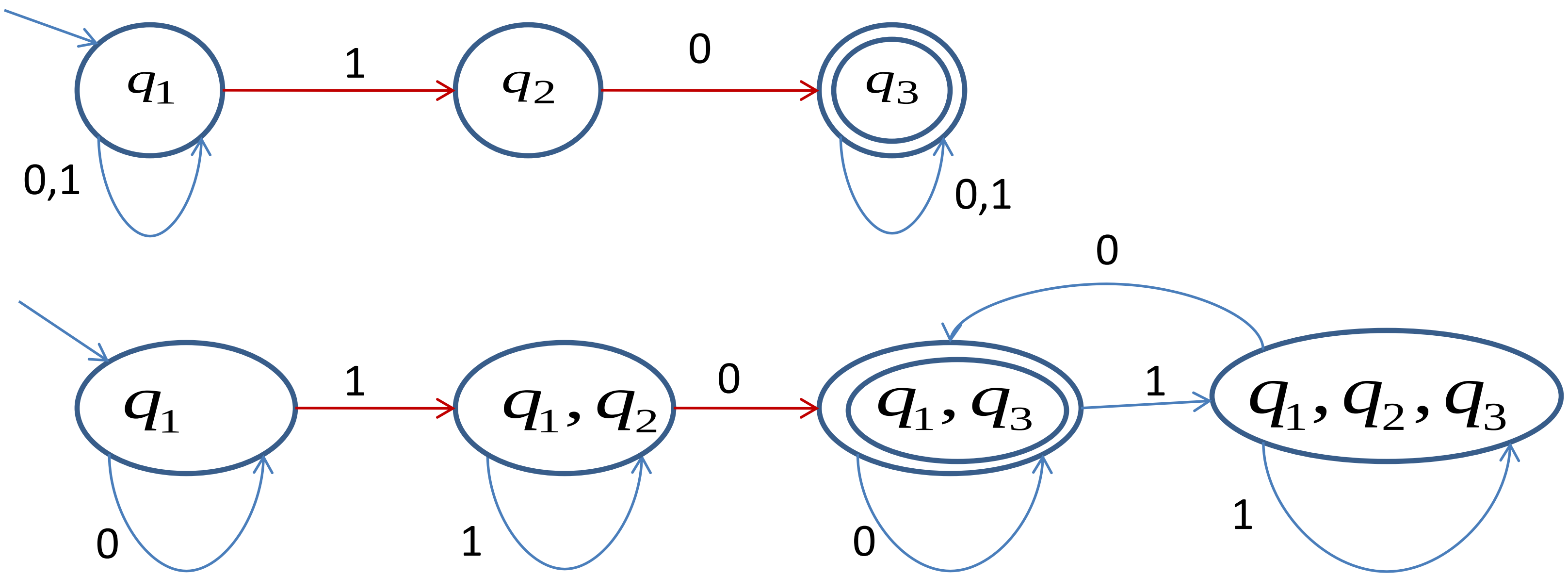
Finally we get the following DFA:

Construction Demonstration



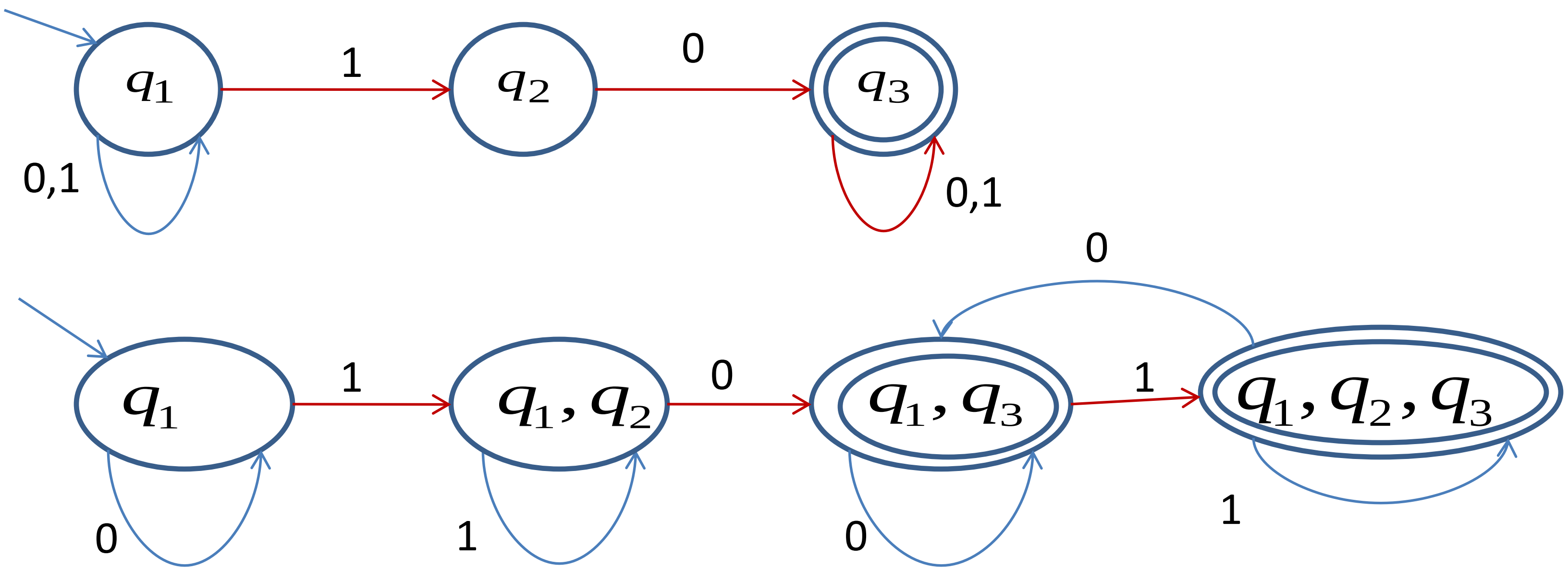
And its disconnected section:

Construction Demonstration



Since 1,0 is accepted we get that $\{q_1, q_3\}$ is an accepting state.

Construction Demonstration



Since 1,0,1 is accepted we get that $\{q_1, q_2, q_3\}$ is also an accepting state.

Nondeterminism

- A *nondeterministic finite automaton* has the ability to be in several states at once.
- Transitions from a state on an input symbol can be to any set of states.

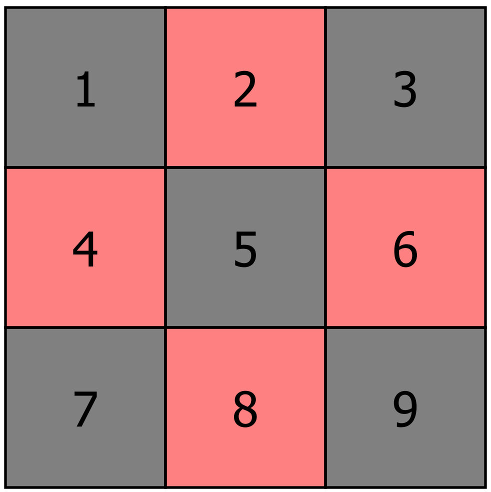
Nondeterminism – (2)

- Start in one start state.
- Accept if any sequence of choices leads to a final state.
- Intuitively: the NFA always "guesses right."

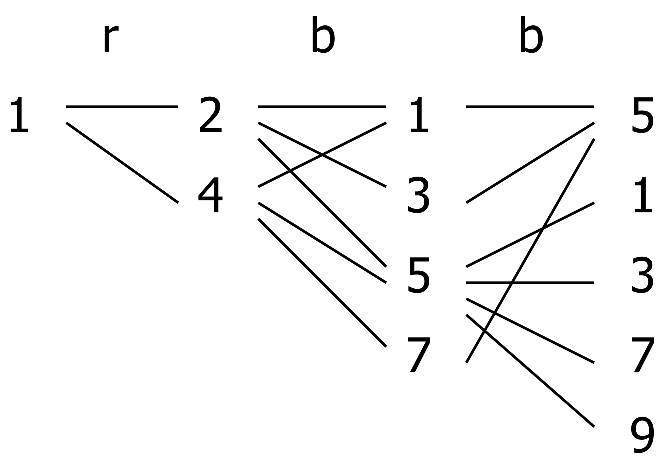
Example: Moves on a Chessboard

- States = squares.
- Inputs = r (move to an adjacent red square) and b (move to an adjacent black square).
- Start state, final state are in opposite corners.

Example: Chessboard – (2)



		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5



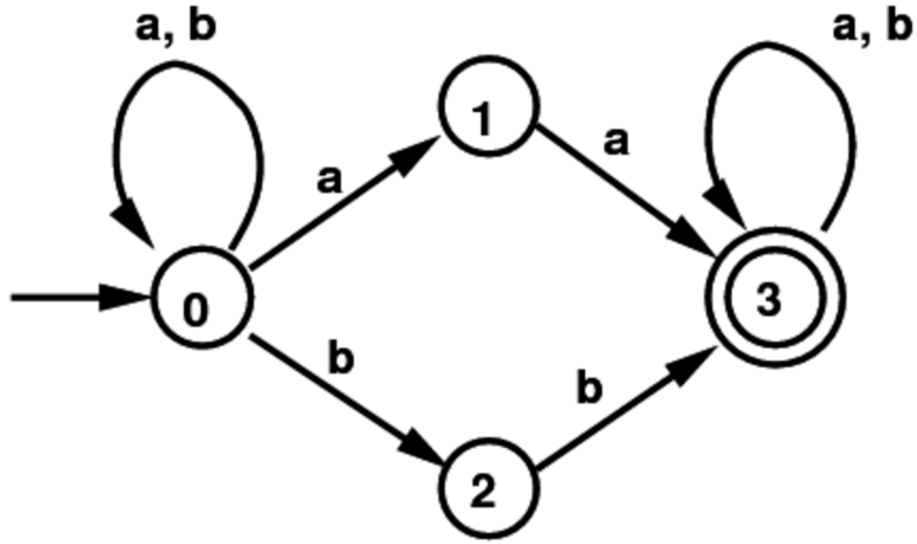
← Accept, since final state reached

Language of an NFA

- A string w is **accepted** by an NFA if $\delta(q_0, w)$ contains at least one final state.
- That is, **there exists** a sequence of valid transitions from q_0 to a final state given the input w.
- The language of the NFA is the set of strings it accepts.

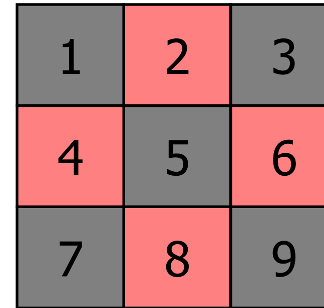
Example NFA

- Set of all strings with two consecutive a’s or two consecutive b’s:



- Note that some states have an empty transition on an a or b, and some have multiple transitions on a or b.

Example 2: Language of an NFA



- For our chessboard NFA we saw that rbb is accepted.
- If the input consists of only b's, the set of accessible states alternates between {5} and {1,3,7,9}, so only even-length, nonempty strings of b's are accepted.
- What about strings with at least one r?

Equivalence of DFA's, NFA's

- A DFA can be turned into an NFA that accepts the same language.
- If $\delta_D(q, a) = p$, let the NFA have $\delta_N(q, a) = \{p\}$.
- Then the NFA is always in a set containing exactly one state – the state the DFA is in after reading the same input.

Equivalence – (2)

- Surprisingly, for any NFA there is a DFA that accepts the same language.
- Proof is the *subset construction*.
- The number of states of the DFA can be exponential in the number of states of the NFA.
- Thus, NFA's accept **exactly** the regular languages.

Subset Construction

- Given an NFA with states Q, inputs Σ, transition function δ_N, state state q_0, and final states F, construct equivalent DFA with:
  – States 2^Q (Set of subsets of Q).
  – Inputs Σ.
  – Start state $\{q_0\}$.
  – Final states = all those with a member of F.

Critical Point

- The DFA states have *names* that are sets of NFA states.
- But as a DFA state, an expression like {p,q} must be read as a single symbol, not as a set.
- Analogy: a class of objects whose values are sets of objects of another class.

Subset Construction – (2)

- The transition function δ_D is defined by:

$\delta_D(\{q_1,\ldots,q_k\}, a)$ is the union over all $i = 1,\ldots,k$ of $\delta_N(q_i, a)$.

- Example: We' ll construct the DFA equivalent of our "chessboard" NFA.

Example: Subset Construction

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

		r	b
→	{1}	{2,4}	{5}
	{2,4}		
	{5}		

Alert: What we're doing here is the *lazy* form of DFA construction, where we only construct a state if we are forced to.

Example: Subset Construction

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}		
	{2,4,6,8}		
	{1,3,5,7}		

Example: Subset Construction

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}		
	{1,3,5,7}		
*	{1,3,7,9}		

Example: Subset Construction

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}	{2,4,6,8}	{1,3,5,7,9}
	{1,3,5,7}		
*	{1,3,7,9}		
*	{1,3,5,7,9}		

Example: Subset Construction

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}	{2,4,6,8}	{1,3,5,7,9}
	{1,3,5,7}	{2,4,6,8}	{1,3,5,7,9}
*	{1,3,7,9}		
*	{1,3,5,7,9}		

Example: Subset Construction

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}	{2,4,6,8}	{1,3,5,7,9}
	{1,3,5,7}	{2,4,6,8}	{1,3,5,7,9}
*	{1,3,7,9}	{2,4,6,8}	{5}
*	{1,3,5,7,9}		

Example: Subset Construction

		r	b
→	1	2,4	5
	2	4,6	1,3,5
	3	2,6	5
	4	2,8	1,5,7
	5	2,4,6,8	1,3,7,9
	6	2,8	3,5,9
	7	4,8	5
	8	4,6	5,7,9
*	9	6,8	5

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}	{2,4,6,8}	{1,3,5,7,9}
	{1,3,5,7}	{2,4,6,8}	{1,3,5,7,9}
*	{1,3,7,9}	{2,4,6,8}	{5}
*	{1,3,5,7,9}	{2,4,6,8}	{1,3,5,7,9}

Proof of Equivalence: Subset Construction

- The proof is almost a pun.
- Show by induction on |w| that

  $\delta_N(q_0, w) = \delta_D(\{q_0\}, w)$

- Basis: w = ϵ: $\delta_N(q_0, \epsilon) = \delta_D(\{q_0\}, \epsilon) = \{q_0\}$.

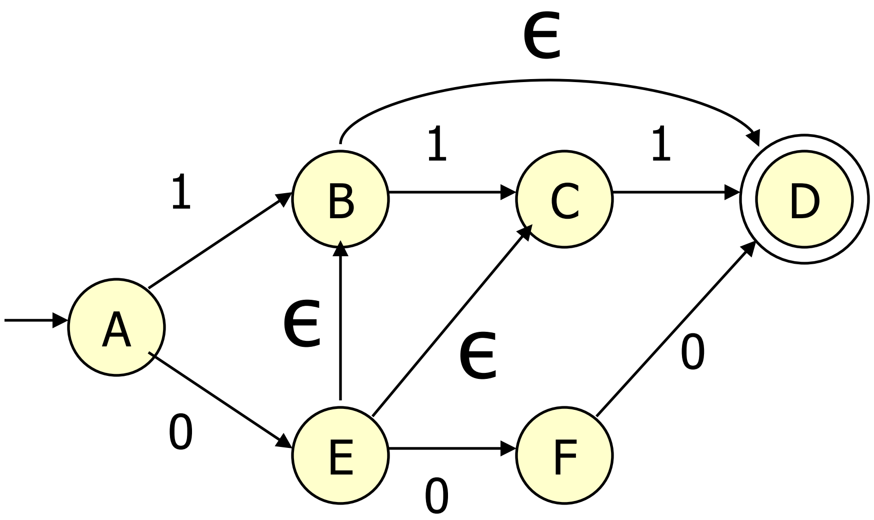
Induction

- Assume Induction Hypothesis for strings shorter than w.
- Let w = xa; Induction H holds for x.
- Let $\delta_N(q_0, x) = \delta_D(\{q_0\}, x) = S$.
- Let T = the union over all states p in S of $\delta_N(p, a)$.
- Then $\delta_N(q_0, w) = \delta_D(\{q_0\}, w) = T$.
  - For NFA: the extension of δ_N.
  - For DFA: definition of δ_D plus extension of δ_D.
    - That is, $\delta_D(S, a) = T$; then extend δ_D to w = xa.

NFA's With ϵ-Transitions

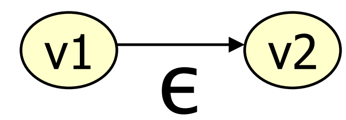
- We can allow state-to-state transitions on ϵ input.
- These transitions are done spontaneously, without looking at the input string.
- A convenience at times, but still only regular languages are accepted.

Example: ϵ-NFA

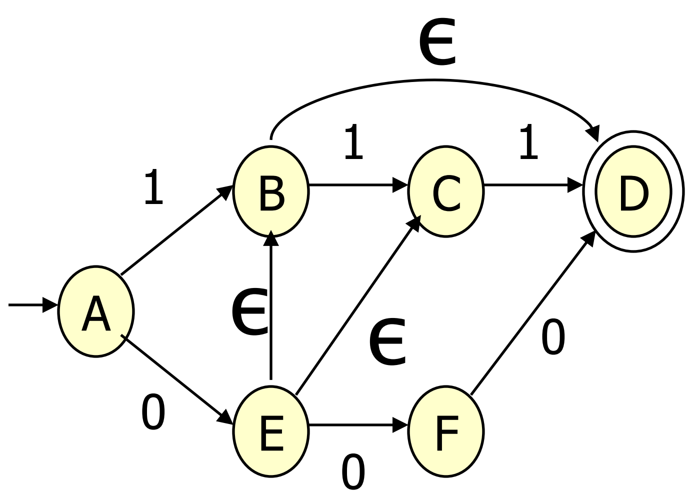


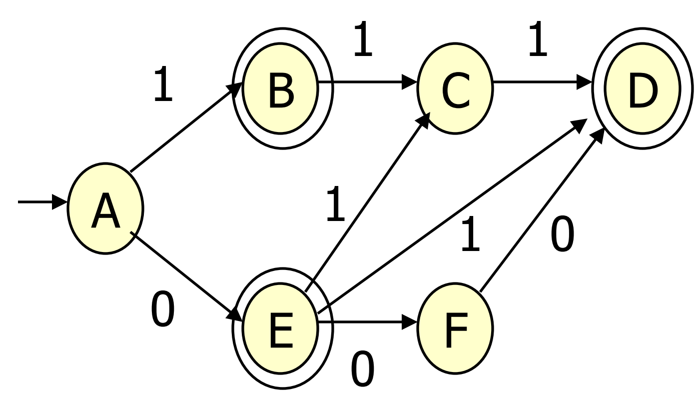
		0	1	ϵ
→	A	{E}	{B}	∅
	B	∅	{C}	{D}
	C	∅	{D}	∅
	D	∅	∅	∅
*	E	{F}	∅	{B, C}
	F	{D}	∅	∅

Conversion of a Transition System with ϵ-move to a TS without ϵ-move



- S1: Find all edges starting from v2
- S2: Duplicate all these edges starting from v1 without changing edge labels.
- S3: If v1 is an initial state, make v2 as an initial state
- S4: If v2 is a final state, make v1 as a final state.





Conversion of a NDFA to a DFA

- S1: Conversion of a TS with ϵ-moves to a TS without ϵ-moves
- S2:Create the start state of DFA by taking the ϵ-closure of the start states of NDFA. Construct a state table for TS
- S3: Construct a Successor Table for the NDFA
  - 3a. Begin with the start state and compute $\delta(q_0, a)$ for every $a \in \sum$; this gives a number of new states Q'
  - 3b. For each new state Q', we compute $\delta(Q', a)$ for every $a \in \sum$ and introduce new states if necessary
  - 3c. Repeat 3b. until there are no new states.
  - 3d. Final states of DFA are the states that contain any final state of NDFA
- S4: Reduce the number of states if possible.
- S5: Draw the state table from the DFA
- Note: Cannot merge non-final and final states in S4.

① ⑧ Start with a set of initial states

Prob Obtain the deterministic graph equivalent to the transition system

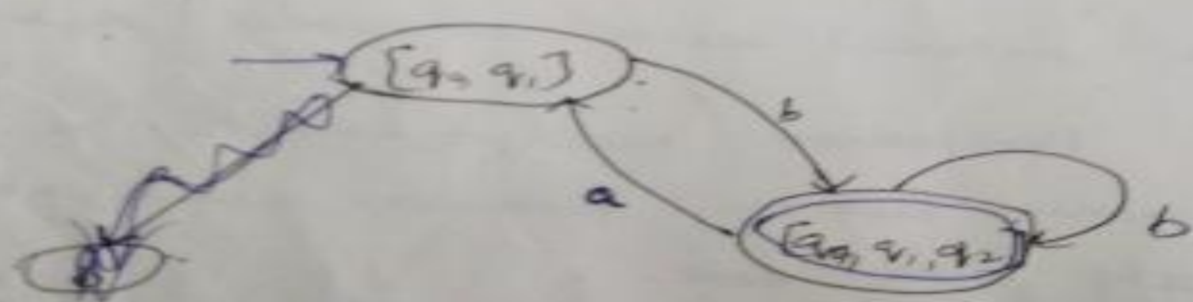
NDFA & DFA

Soln

state / Σ	a	b
→ q_0		$\{q_1, q_2\}$
→ q_1		$\{q_2\}$
q_2 *	$\{q_0, q_1\}$	

Construct successor table

Q	a	b
→ $[q_0, q_1]$	ϕ	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2]$	$[q_0, q_1]$	$[q_0, q_1, q_2]$
ϕ	ϕ	ϕ



S1

Q \ Σ	0	1
→ q_0	$\{q_2\}$	$\{q_1, q_4\}$
q_1	—	$\{q_0\}$
q_2	$\{q_3\}$	—
→ q_3	$\{q_2\}$	$\{q_4\}$
q_4	$\{q_5\}$	—
q_5	[illegible]	{q6}

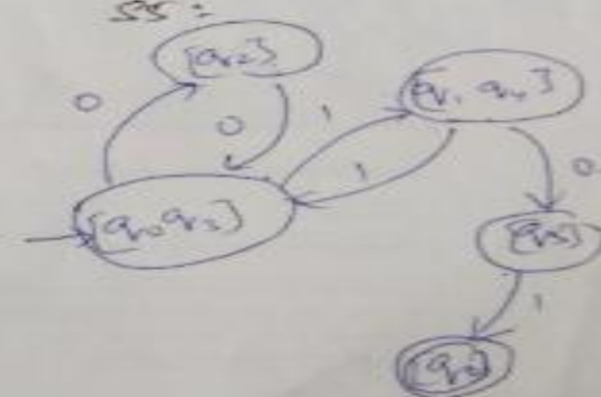
Successor Table

S2

Q \ Σ	0	1
$[q_0, q_3]$	$[q_2]$	$[q_1, q_4]$
$[q_2]$	$[q_0, q_3]$	—
$[q_1, q_4]$	$[q_5]$	$[q_0, q_2]$
$[q_5]$	—	$[q_6]$
$[q_6]$	—	—

S3: x S4: x

S5:



Summary

- DFA' s, NFA' s, and ϵ–NFA' s all accept exactly the same set of languages: the regular languages.
- The NFA types are easier to design and may have exponentially fewer states than a DFA.
- But only a DFA can be implemented in linear time!

General Comments

- Some things are easy with finite automata:
  - Substrings (…abcabc…)
  - Subsequences (…a…b…c…b…a…)
  - Modular counting (odd number of 1's)
- Some things are impossible with finite automata :
  - An equal number of a's and b's
  - More 0's than 1's
- But when they **can** be used, they are fast.

Summary

- RE to RS, RE to RL
- RS to RL
- NFA to DFA
- NFA/DFA to RL

- RE to NFA ? Next slide
- DFA to RE ? Arden's Theorem not covered

RE to NFA Kleene's Theorem

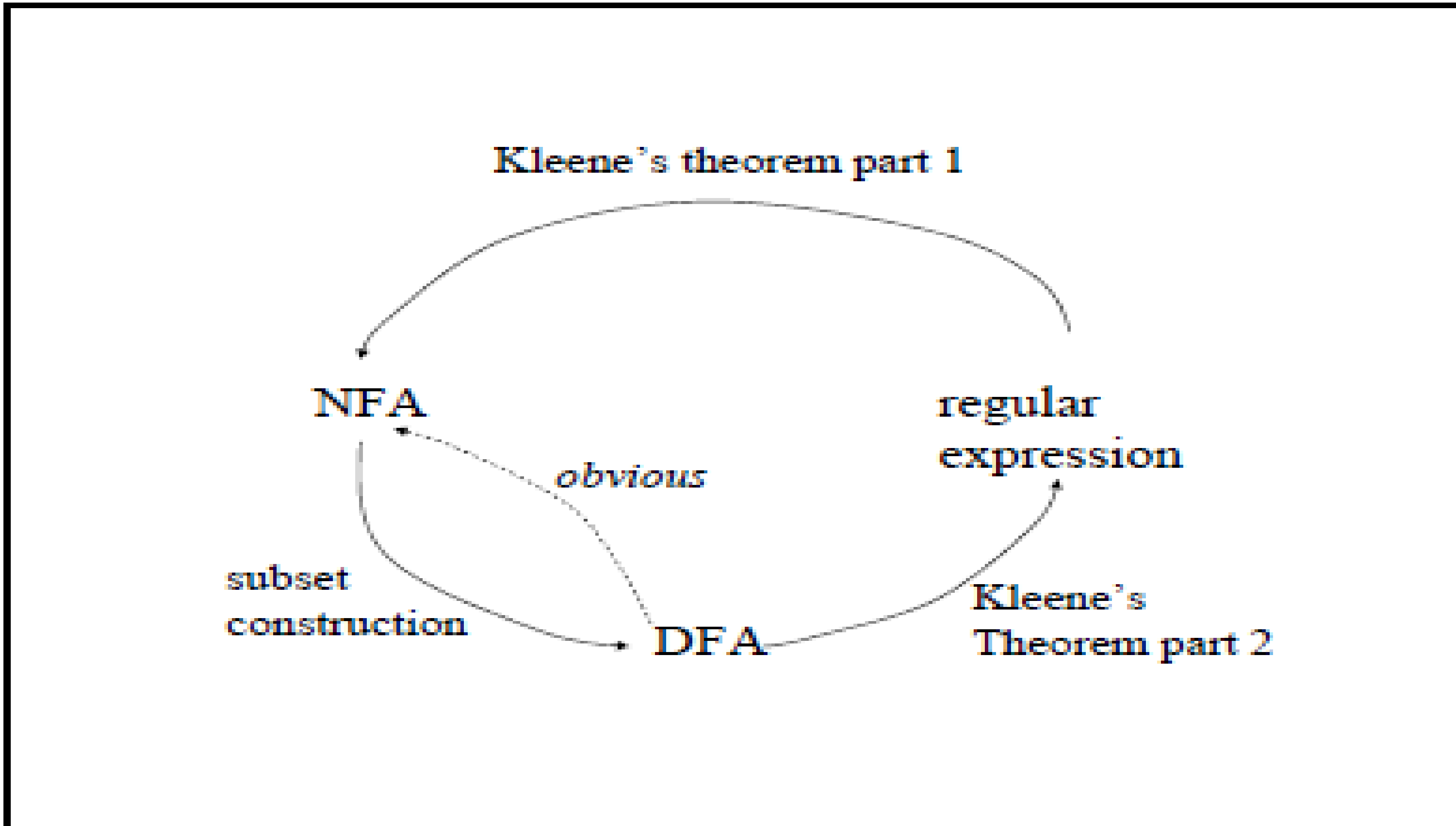
Kleene's theorem

1) For any regular expression r that represents language L(r), there is a finite automaton that accepts that same language.

2) For any finite automaton M that accepts language L(M), there is a regular expression that represents the same language.

Therefore, the class of languages that can be represented by regular expressions is equivalent to the class of languages accepted by finite automata -- the *regular languages*.

RE to NFA Kleene's Theorem



RE to NFA Kleene’s Theorem

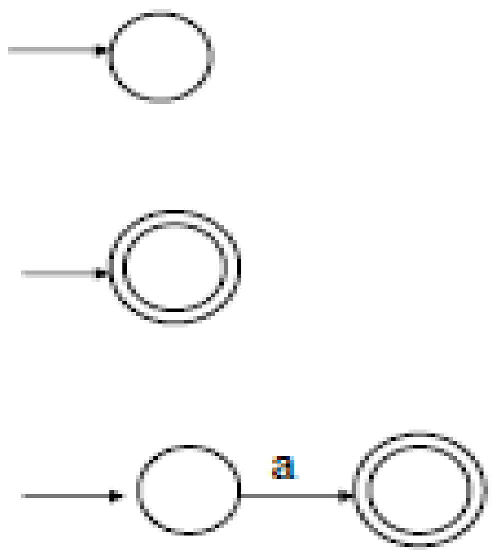
Proof of 1st half of Kleene’s theorem

Proof strategy: for any regular expression,
we show how to construct an equivalent NFA.

Because regular expressions are defined recursively,
the proof is by induction.

RE to NFA Kleene's Theorem

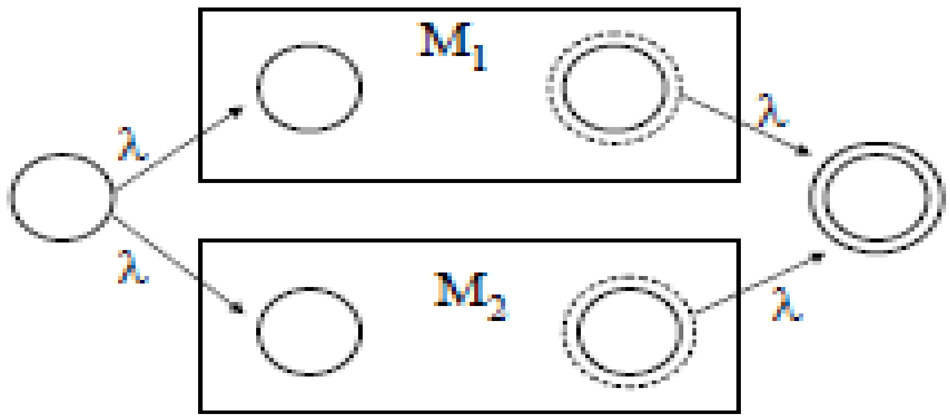
Base step: Give a NFA that accepts each of the simple or "base" languages, $\emptyset$, $\{\lambda\}$, and $\{a\}$ for each $a \in \Sigma$.



RE to NFA Kleene's Theorem

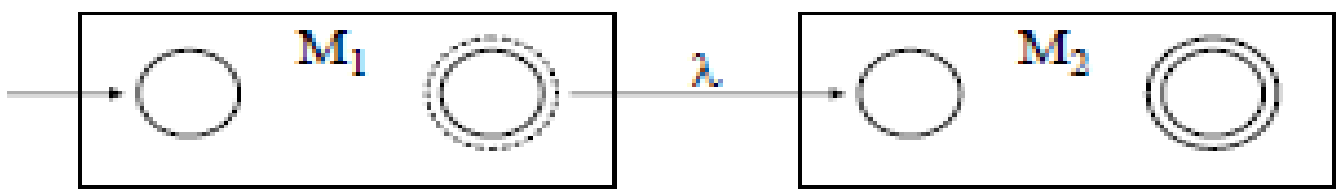
Inductive step: For each of the operations -- union, concatenation and Kleene star -- show how to construct an accepting NFA.

Closure under union:

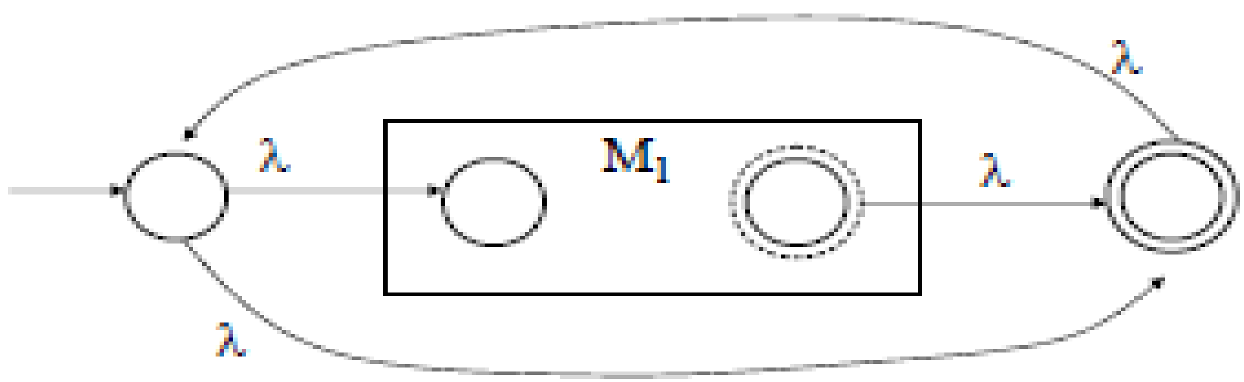


RE to NFA Kleene’s Theorem

Closure under concatenation:



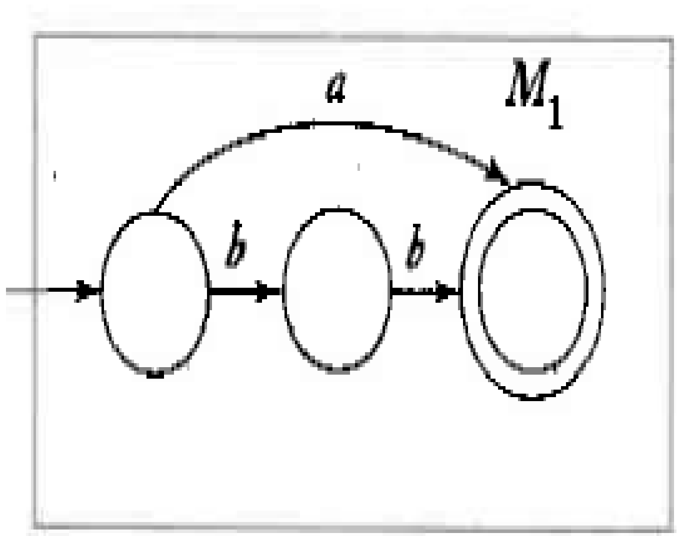
Closure under Kleene Star:

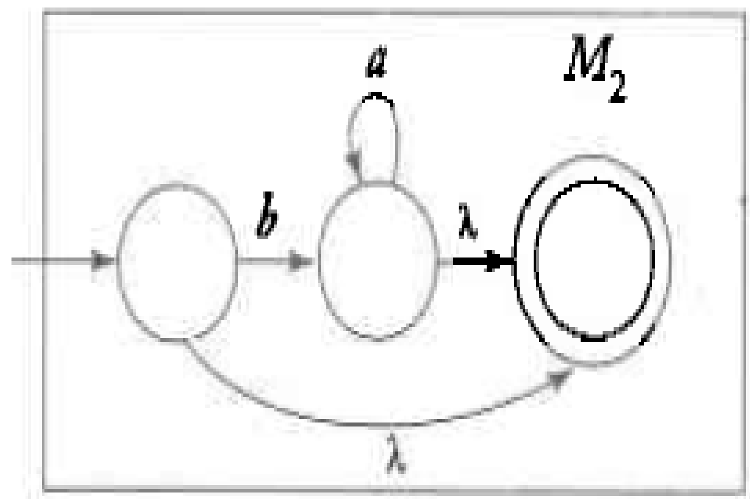


Examples

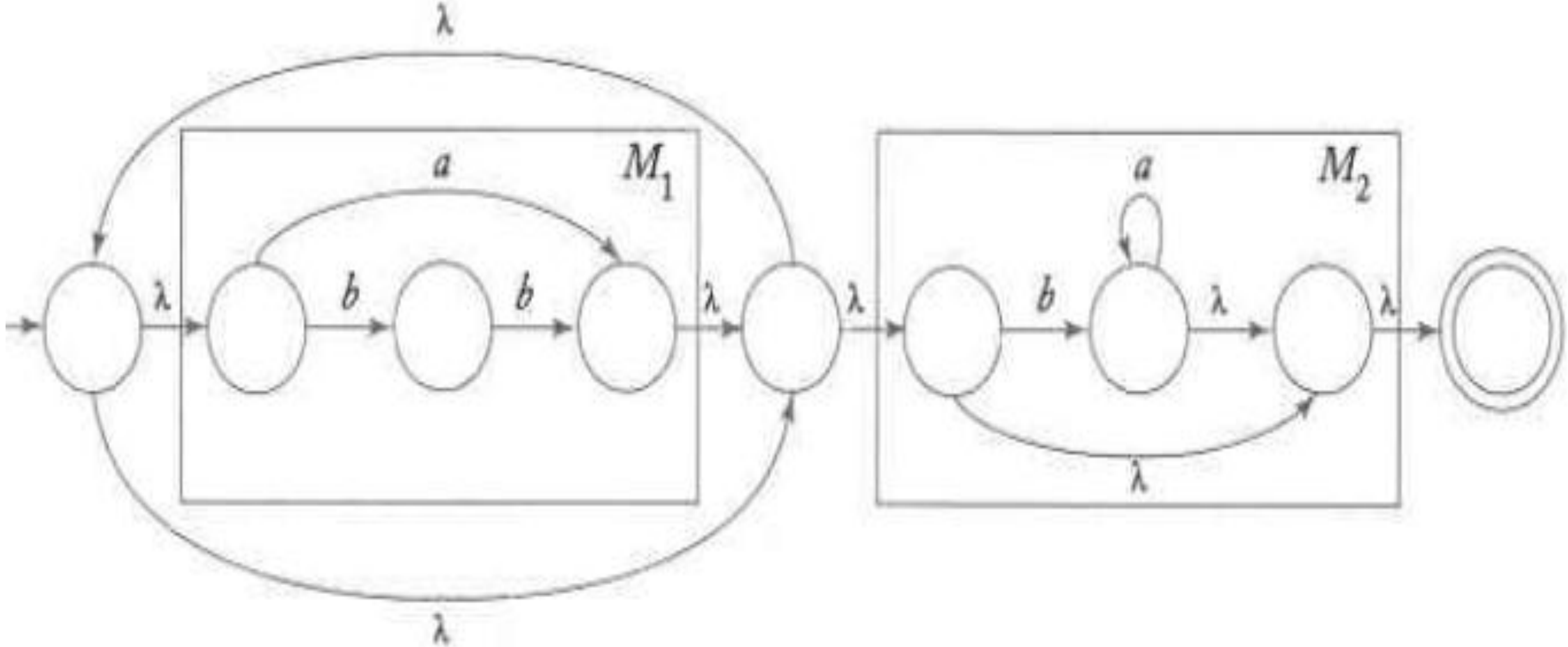
- Eg1: (a+bb)

Eg2:(λ+ba*)





Example: (a+bb)*(λ+ba*)



$$r = r_1^* r_2 \left(r_4 + r_3 r_1^* r_2\right)^*$$