Welcome

FORMAL LANGUAGES

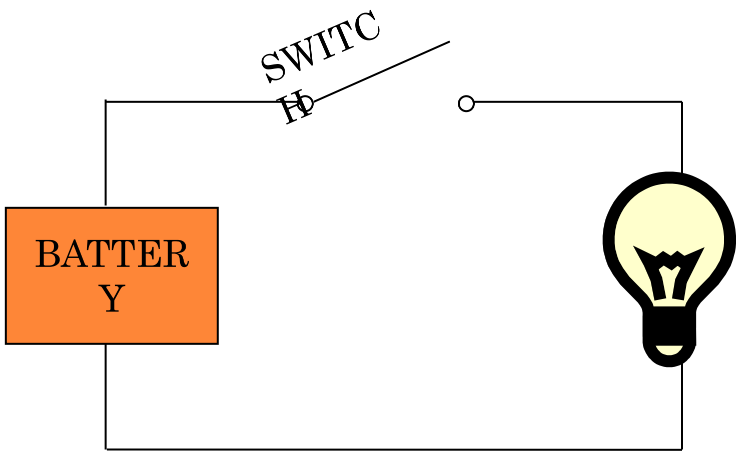
AND

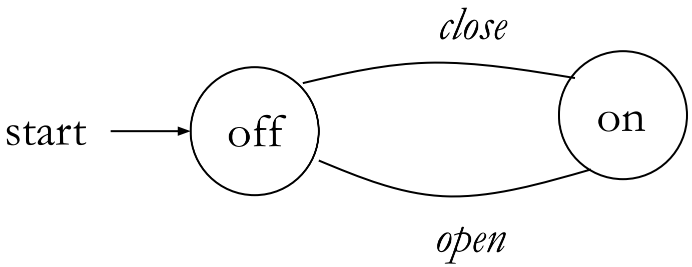
AUTOMATA THEORY

What is automata theory

- Automata theory is the study of abstract computational devices
- Abstract devices are (simplified) models of real computations
- Computations happen everywhere: On your laptop, on your cell phone, in nature, …
- Why do we need abstract models?

A simple "Automata"





input: switch

output: light bulb

actions: open and close

states: on, off

bulb is on if and only if the switch is closed.

Brief History

1930s	• Alan Turing studies Turing machines • Decidability • Halting problem
1940-1950s	• "Finite automata" machines studied • Noam Chomsky proposes the "Chomsky Hierarchy" for formal languages
1969	Cook introduces "intractable" problems or "NP-Hard" problems
1970-	Modern computer science: compilers, computational & complexity theory evolve

Applications

- For designing and checking the behavior of digital circuits.
- Lexical analyzer of a typical compiler.
- For scanning large bodies of text (e.g., web pages) for pattern matching.
- For verifying systems of all types that have a finite number of states (e.g., stock market transaction, communication/network protocol).
- NLP (Natural Language Processing).

Alphabet

- A finite, nonempty set of symbols.
- Symbol: Σ
- Examples:
  - The binary alphabet: $\Sigma = \{0, 1\}$
  - The set of all lower-case letters: $\Sigma = \{a, b, \ldots, z\}$
  - The set of all ASCII characters
  - The set of alphanumeric: $\sum = \{a\text{-}z, A\text{-}Z, 0\text{-}9\}$
  - The set of vowels : $\sum = \{a,e,i,o,u\}$

Strings

- A **string** (or sometimes a **word**) is a finite sequence of symbols chosen from some alphabet **Σ.**
- Example: 01101 and 111 are strings from the binary alphabet Σ = {0, 1}
- **Empty string**: the string with zero occurrences of symbols

  This string is denoted by ε and may be chosen from any alphabet whatsoever.
- **Length of a string**: the number of positions for symbols in the string

  Example: 01101 has length 5

  - There are only two symbols (0 and 1) in the string 01101, but 5 positions for symbols.
- Notation of length of *w*: |*w*|

  Example: |011| = 3 and | ε | = 0

Powers of an alphabet

If Σ is an alphabet, we can express the set of all strings of a certain length from that alphabet by using the exponential notation:

- Σ^k: the set of strings of length k, each of whose is in Σ
- Examples:
  - Σ^0 : $\{\varepsilon\}$, regardless of what alphabet Σ is. That is ε is the only string of length 0
  - If $\Sigma = \{0, 1\}$, then:

    1. $\Sigma^1 = \{0, 1\}$
    2. $\Sigma^2 = \{00, 01, 10, 11\}$
    3. $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

    Note: confusion between Σ and Σ^1:

    1. Σ is an alphabet; its members 0 and 1 are symbols
    2. Σ^1 is a set of strings; its members are strings (each one of length 1)

Kleen star

- Σ^*: The set of all strings over an alphabet Σ
  - $\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$
  - $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- The symbol $*$ is called **Kleene star** and is named after the mathematician and logician Stephen Cole Kleene.
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots$

  Thus: $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$

Languages

- If Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a (formal) **language** over Σ.
- Language: A (possibly infinite) set of strings all of which are chosen from some Σ^*.
- A language over Σ need not include strings with all symbols of Σ.

  Thus, a language over Σ is also a language over any alphabet that is a superset of Σ.
- Examples:
  - Programming language C

    Legal programs are a subset of the possible strings that can be formed from the alphabet of the language (a subset of ASCII characters)
  - English or Telugu.

Other language examples

1. The language of all strings consisting of n 0s followed by n 1s ($n \geq 0$):

   $\{\varepsilon, 01, 0011, 000111, \ldots\}$

2. The set of strings of 0s and 1s with an equal number of each:

   $\{\varepsilon, 01, 10, 0011, 0101, 1001, \ldots\}$

3. Σ^* is a language for any alphabet Σ
4. $\varnothing$, the empty language, is a language over any alphabet.
5. $\{\varepsilon\}$, the language consisting of only the empty string, is also a language over any alphabet.

   NOTE: $\varnothing \ \{\varepsilon\} = \{\varepsilon\}$ since $\varnothing$ has no strings and $\{\varepsilon\}$ has one

6. $\{w \mid w$ consists of an equal number of 0 and 1$\}$
7. $\{0^n 1^n \mid n \geq 1\}$
8. $\{0^i 1^j \mid 0 \leq i \leq j\}$

OPERATORS ON LANGUAGES: UNION

The **union** of two languages L and M, denoted $L \cup M$, is the set of strings that are in either L, or M, or both.

Example If $L = \{001, 10, 111\}$ and $M = \{\varepsilon, 001\}$ then

$$L \cup M = \{\varepsilon, 001, 10, 111\}$$

Operators on languages: Concatenation

The **concatenation** of languages L and M, denoted $L.M$ or just LM, is the set of strings that can be formed by taking any string in L and concatenating it with any string in M.

Example If $L = \{001, 10, 111\}$ and $M = \{\varepsilon, 001\}$ then

$$L.M = \{001, 10, 111, 001001, 10001, 111001\}$$

OPERATORS ON LANGUAGES: CLOSURE

The **closure** of a language L is denoted L^* and represents the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions (*i.e.*, the same string may be selected more than once) and concatenating all of them.

Examples:

- If L = {0, 1} then L^* is all strings of 0 and 1
- If L = {0, 11} then L^* consists of strings of 0 and 1 such that the 1 comes in pair, *e.g.*, 011, 11110 and ε. But not 01011 or 101.

Formally, L^* is the infinite union $\bigcup_{i \geq 0} L^i$ where $L^0 = \{\varepsilon\}$, $L^1 = L$, and for $i > 1$

Regular Expression

- A compact or mathematical representation of a regular language is known as regular expression.

Regular Expressions and Languages

We define the regular expressions recursively.

Basis: The basis consists of three parts:

1. The constants **ε** and $\varnothing$ are regular expressions, denoting the language $\{\varepsilon\}$ and $\varnothing$, respectively. That is $L(\varepsilon) = \{\varepsilon\}$ and $L(\varnothing) = \varnothing$.
2. If a is a symbol, then **a** is a regular expression. This expression denotes the language $\{a\}$, *i.e.*, $L(\mathbf{a}) = \{a\}$.

   NOTE: We use boldface font to denote an expression corresponding to a symbol
3. A variable, usually capitalised and italic such as L, is a variable, representing any language.

Regular Expressions and Languages

Induction: There are four parts to the inductive step, one for each of the three operators and one for the introduction of parentheses

1. If E and F are regular expressions, then $E + F$ is a regular expression denoting the union of $L(E)$ and $L(F)$. That is, $L(E + F) = L(E) \cup L(F)$.
2. If E and F are regular expressions, then EF is a regular expression denoting the concatenation of $L(E)$ and $L(F)$. That is, $L(EF) = L(E)L(F)$.
3. If E is a regular expression, then E^* is a regular expression denoting the closure of $L(E)$. That is, $L(E^*) = (L(E))^*$.
4. If E is a regular expression, then (E) is a regular expression denoting the same as E. Formally, $L((E)) = (L(E))$.

Finite Automaton (FA)

- Informally, FA is a state diagram that consists of states and transitions.
- Recognizer for "Regular Languages"
- Deterministic Finite Automata (DFA)
  - The machine can exist in only one state at any given time
- Non-deterministic Finite Automata (NFA)
  - The machine can exist in multiple states at the same time

Formal Definition of a DFA

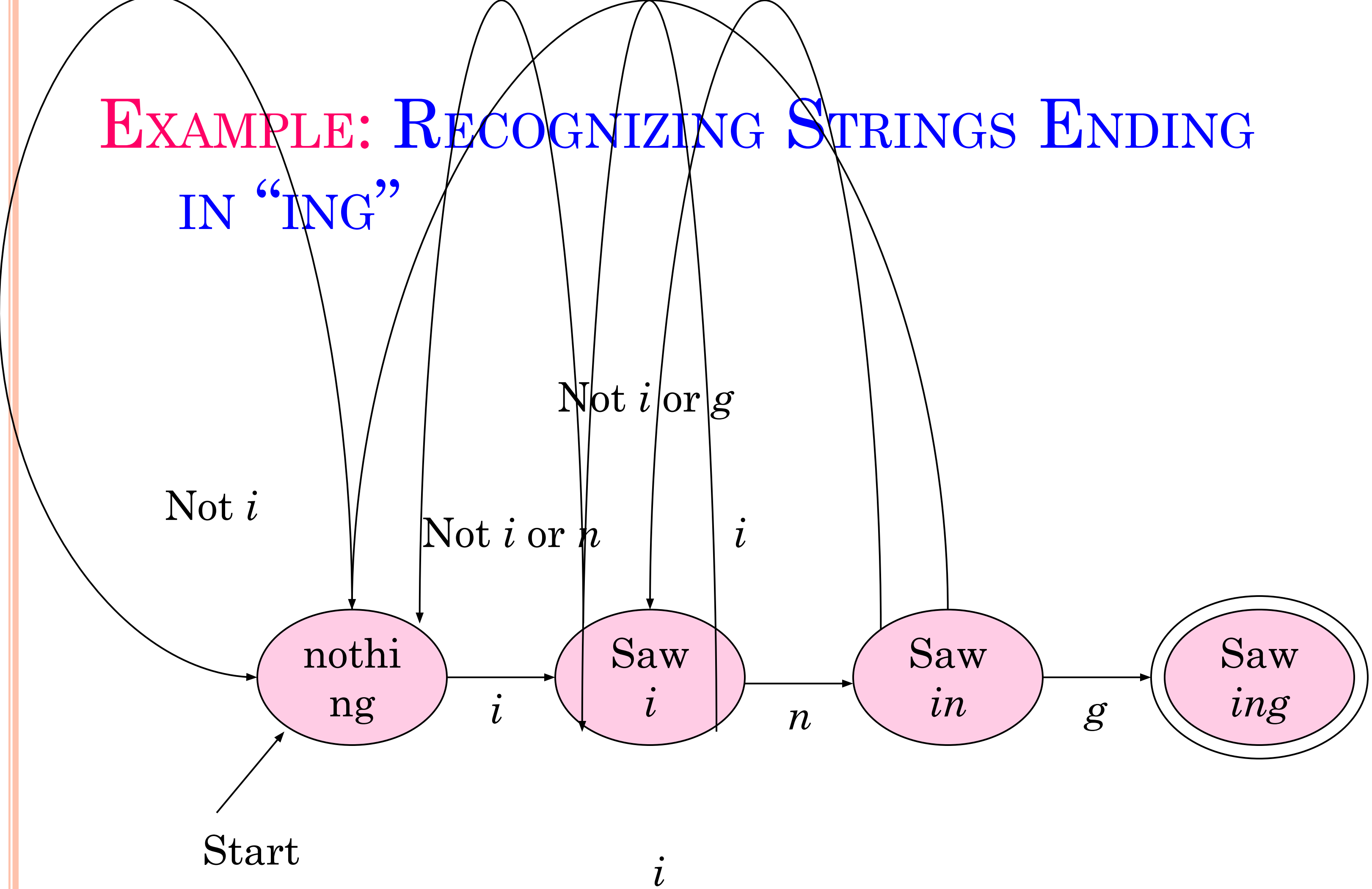
- A DFA is a five-tuple: $M = (Q, \Sigma, \delta, q_0, F)$

Where:

Q	A finite set of states
Σ	A finite input alphabet
q_0	The initial/starting state, q_0 is in Q
F	A set of final/accepting states, which is a subset of Q
δ	A transition function, which is a total function from $Q \times \Sigma$ to Q

$$\delta: (Q \times \Sigma) \rightarrow Q$$

Example: Recognizing Strings Ending in "ing"



- Example #1: Design DFA to accept string contain even number of 0's over an alphabet $\Sigma = \{0, 1\}$.

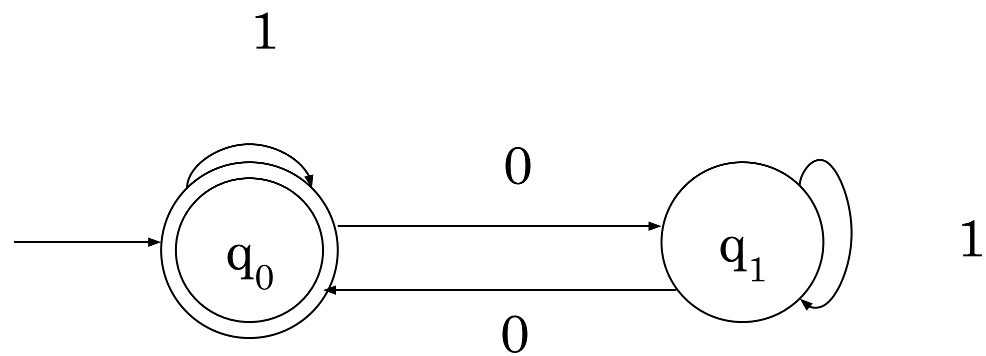
- $L = \{\varepsilon, 00, 0000, 000000, \ldots.\}$

$Q = \{q_0, q_1\}$

$\Sigma = \{0, 1\}$

Start state is q_0

$F = \{q_0\}$



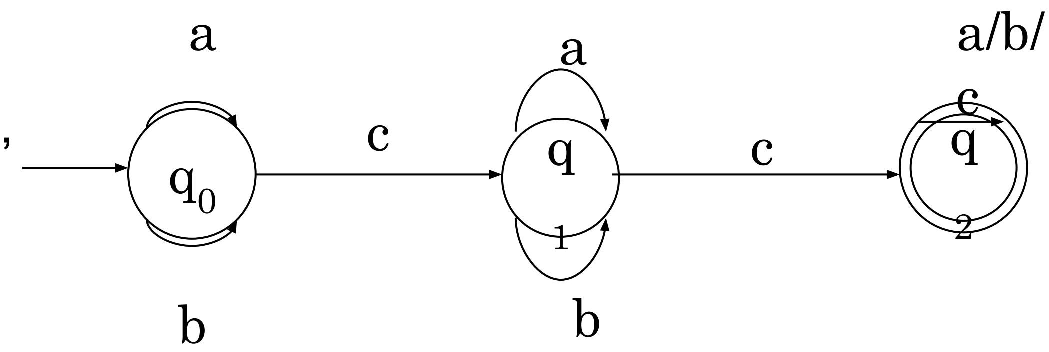
δ:

	0	1	
q_0	q_1	q_0	
q_1	q_0	q_1	

- Example #2: Design DFA to accept at least two ‘c’ symbols in a given string over an alphabet Σ = {a, b, c}.

Sol:

L={cc, acc, bcc, ccc, cac, cbc, …..}



$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{a, b, c\}$

Start state is q_0

$F = \{q_2\}$

δ:	a	b	c
q_0	q_0	q_0	q_1
q_1	q_1	q_1	q_2
q_2	q_2	q_2	q_2

- Since δ is a function, at each step M has exactly one option.
- It follows that for a given string, there is exactly one computation.

Extension of δ to Strings

$\delta^{\wedge} : (Q \times \Sigma^{*}) \rightarrow Q$

$\delta^{\wedge}(q, w)$ – The state entered after reading string *w* having started in state *q*.

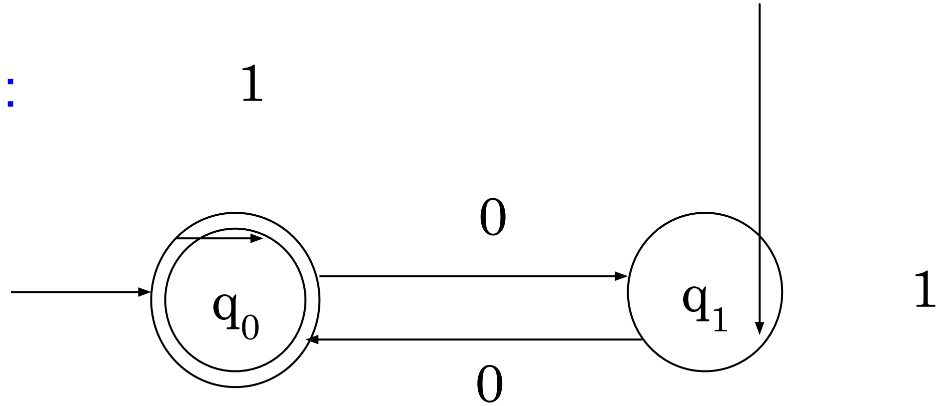
Formally:

1) $\delta^{\wedge}(q, \varepsilon) = q$, and

2) For all w in Σ^{*} and a in Σ

$$\delta^{\wedge}(q,wa) = \delta(\delta^{\wedge}(q,w), a)$$

- Recall Example #1:



- What is $\delta^{\wedge}(q_0, 011)$? Informally, it is the state entered by M after processing 011 having started in state $q_{0.}$
- Formally:

$$
\begin{aligned}
\delta^{\wedge}(q_0, 011) &= \delta\,(\delta^{\wedge}(q_0,01), 1) && \text{by rule \#2} \\
&= \delta\,(\delta\,(\,\delta^{\wedge}(q_0,0), 1), 1) && \text{by rule \#2} \\
&= \delta\,(\delta\,(\delta\,(\delta^{\wedge}(q_0, \lambda), 0), 1), 1) && \text{by rule \#2} \\
&= \delta\,(\delta\,(\delta(q_0,0), 1), 1) && \text{by rule \#1} \\
&= \delta\,(\delta\,(q_1, 1), 1) && \text{by definition of } \delta \\
&= \delta\,(q_1, 1) && \text{by definition of } \delta \\
&= q_1 && \text{by definition of } \delta
\end{aligned}
$$

- Is 011 accepted? No, since $\delta^{\wedge}(q_0, 011) = q_1$ is not a final state.

- Note that:

  $\delta^{\wedge}(q,a) = \delta(\delta^{\wedge}(q, \varepsilon), a)$ by definition of $\delta^{\wedge}$, rule #2

  $= \delta(q, a)$ by definition of $\delta^{\wedge}$, rule #1

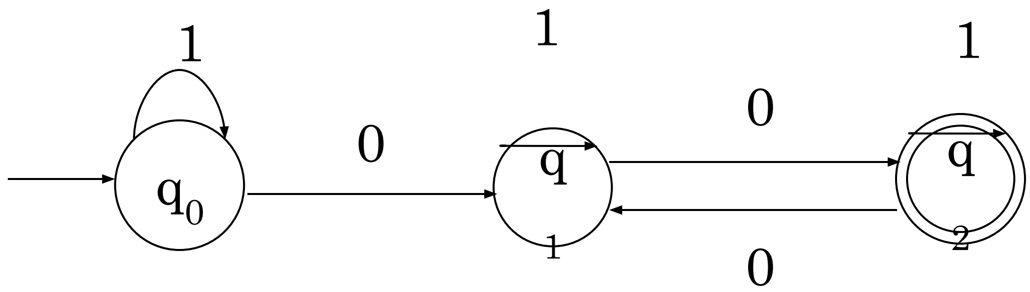
- Therefore:

  $\delta^{\wedge}(q, a_1a_2\ldots a_n) = \delta(\delta(\ldots\delta(\delta(q, a1), a2)\ldots), a_n)$

- However, we will abuse notations, and use δ in place of $\delta^{\wedge}$:

  $\delta^{\wedge}(q, a_1a_2\ldots a_n) = \delta(q, a_1a_2\ldots a_n)$

- **Example #3:** What is δ(q0, 011)? Informally, it is the state entered by M after processing 011 having started in state q0.

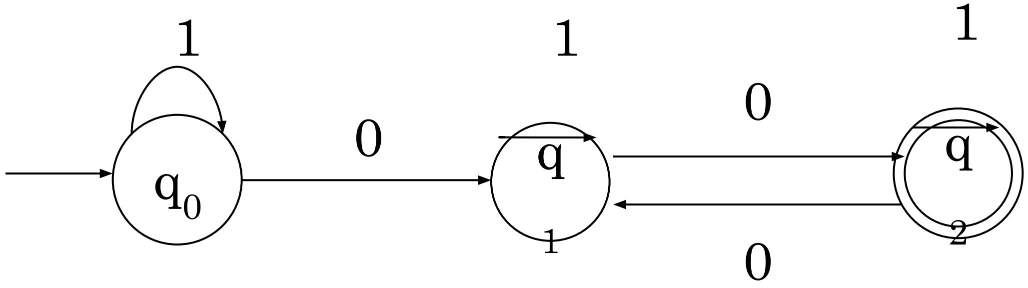


- Formally:

$$\delta(q_0, 011) = \delta(\delta(q_0,01), 1) \quad \text{by rule \#2}$$
$$= \delta(\delta(\delta(q_0,0), 1), 1) \quad \text{by rule \#2}$$
$$= \delta(\delta(q_1, 1), 1) \quad \text{by definition of } \delta$$
$$= \delta(q_1, 1) \quad \text{by definition of } \delta$$
$$= q_1 \quad \text{by definition of } \delta$$

- Is 011 accepted? No, since $\delta(q_0, 011) = q_1$ is not a final state.
- *Language?*
- L ={ all strings over {0,1} that has even number of *0* symbols}

- Recall Example #3:



- What is $\delta(q_1, 10)$?

$\delta(q_1, 10) = \delta(\delta(q_1, 1), 0)$ by rule #2

$= \delta(q_1, 0)$ by definition of δ

$= q_2$ by definition of δ

- Is 10 accepted? No, since $\delta(q_0, 10) = q_1$ is not a final state. The fact that $\delta(q_1, 10) = q_2$ is irrelevant, q1 is not the start state!

Definitions related to DFAs

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let w be in Σ^*. Then w is *accepted* by M iff $\delta(q_0, w) = p$ for some state p in F.

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Then the *language accepted* by M is the set:

  $L(M) = \{w \mid w \text{ is in } \Sigma^* \text{ and } \delta(q_0, w) \text{ is in } F\}$

- Another equivalent definition:

  $L(M) = \{w \mid w \text{ is in } \Sigma^* \text{ and } w \text{ is accepted by } M\}$

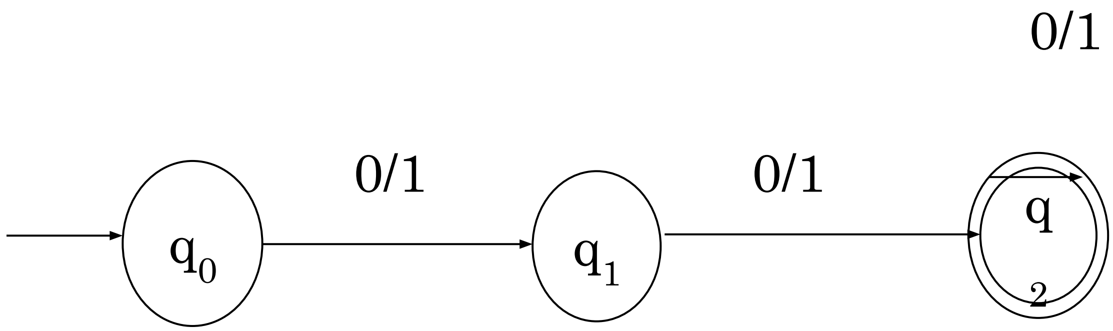
- Let L be a language. Then L is a ***regular language*** iff there exists a DFA M such that $L = L(M)$.

- Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_0, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, p_0, F_2)$ be DFAs. Then M_1 and M_2 are *equivalent* iff $L(M_1) = L(M_2)$.

- Example #4: Design DFA to accept all strings whose length is greater than or equal to 2 over an alphabet Σ = {0, 1}.

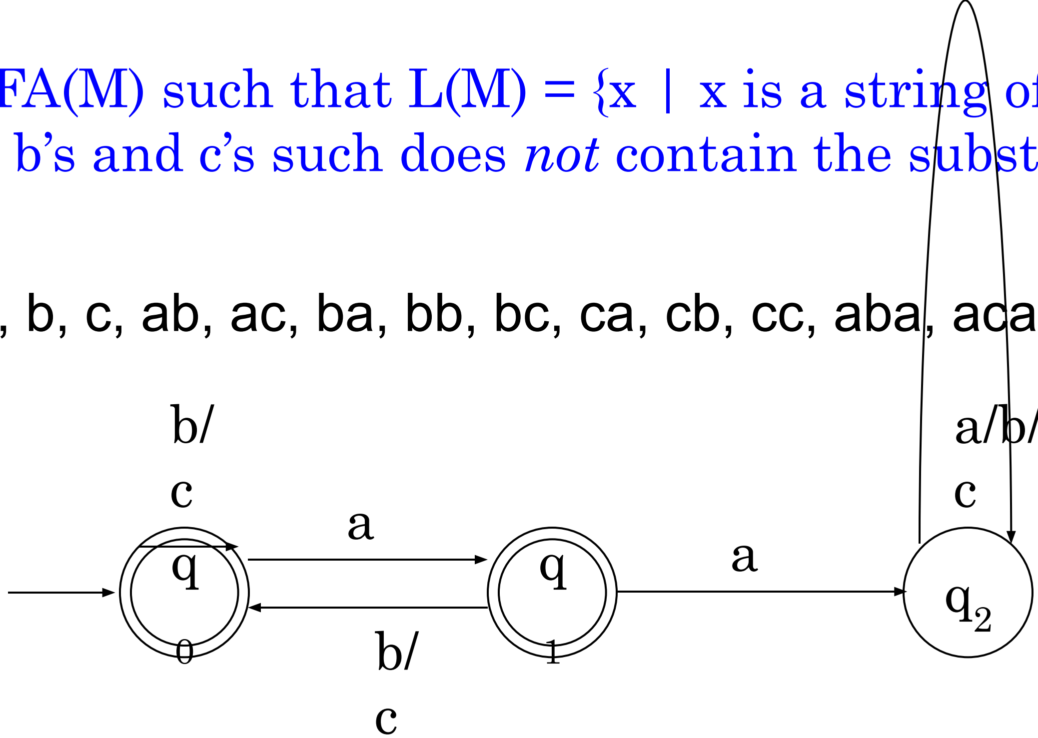
L(M) = {x | x is a string of 0's and 1's and |x| >= 2}

L={00,01,10,11,000,001,010,……..}



- Design DFA(M) such that L(M) = {x | x is a string of (zero or more) a's, b's and c's such does *not* contain the substring *aa*}

Sol: L= {ε, a, b, c, ab, ac, ba, bb, bc, ca, cb, cc, aba, aca, abb, …..}



Logic:

In Start state (q_0): b's and c's: ignore – stay in same state

q_0 is also "accept" state

First 'a' appears: get ready (q_1) to reject

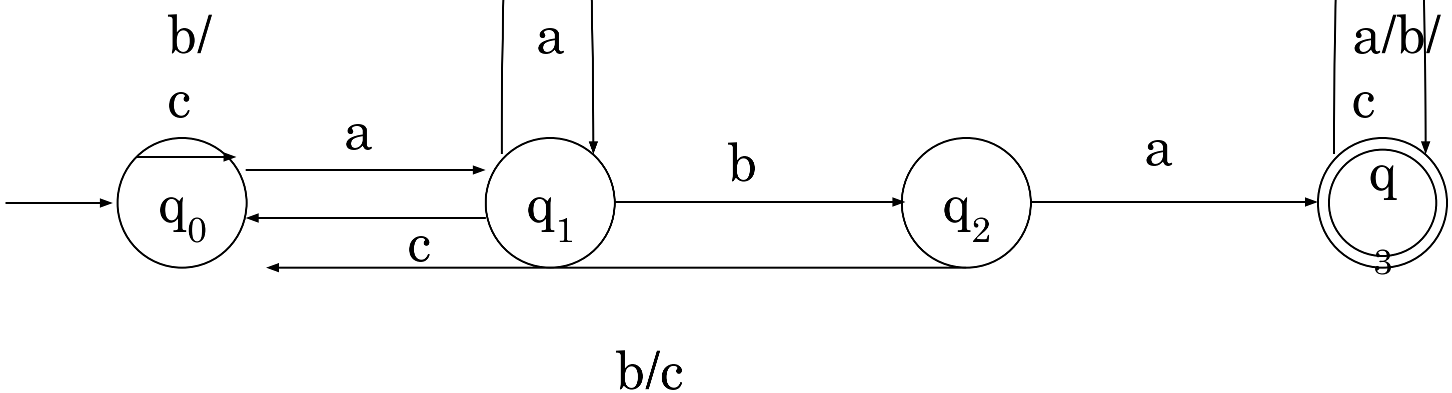
But followed by a 'b' or 'c': go back to start state q0

When second 'a' appears after the "ready" state: go to reject state q2

Ignore everything after getting to the "reject" state q_2

- Give a DFA M such that:

L(M) = {x | x is a string of a's, b's and c's such that x contains the substring *aba*}



Logic: acceptance is straight forward, progressing on each expected symbol

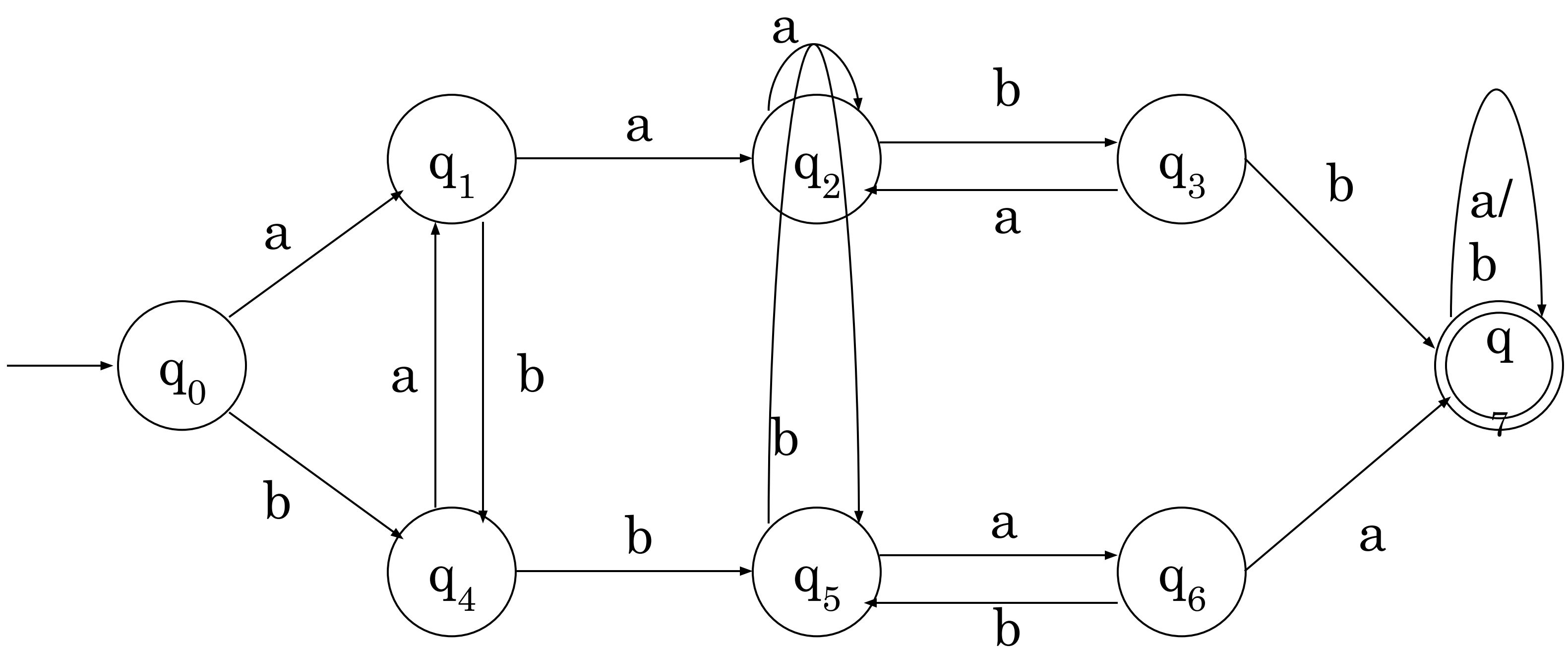
However, rejection needs special care, in each state (for DFA, we will see this becomes easier in NFA, non-deterministic machine)

- Give a DFA M such that:

L(M) = {x | x is a string of a's and b's such that x contains both *aa* and *bb*}

First do, for a language where 'aa' comes before 'bb'

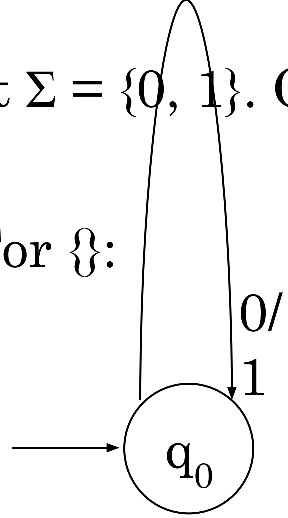
Then do its reverse; and then parallelize them.



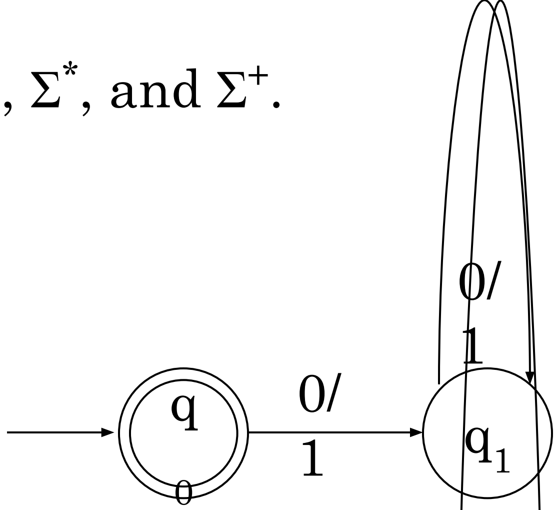
Remember, you may have multiple "final" states, but only one "start" state

- Let $\Sigma = \{0, 1\}$. Give DFAs for $\{\}$, $\{\varepsilon\}$, Σ^*, and Σ^+.

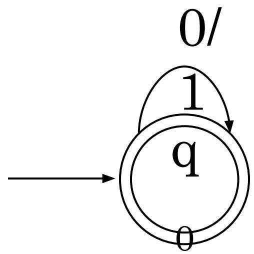
For $\{\}$:



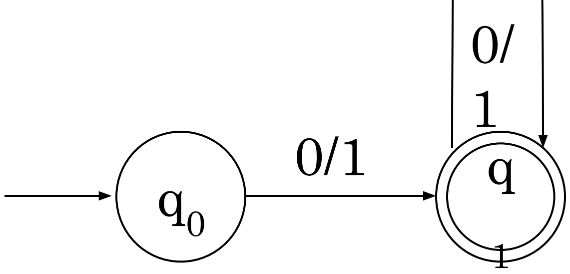
For $\{\varepsilon\}$:



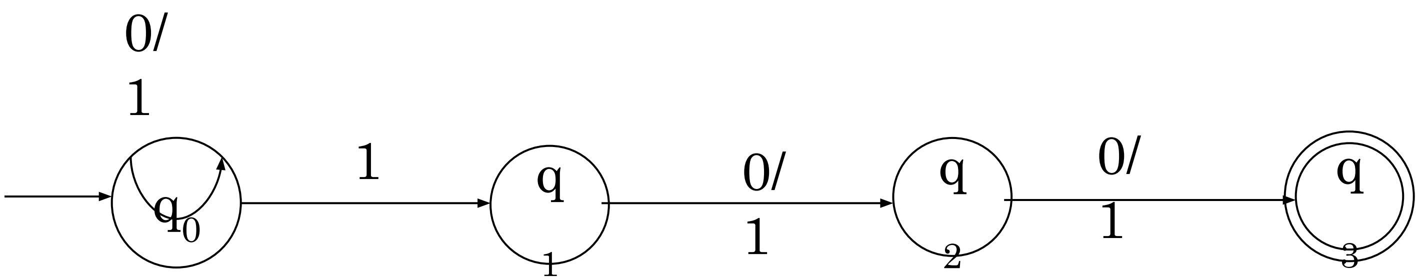
For Σ^*:



For Σ^+:



- Problem: Third symbol from last is *1*



Is this a
DFA?

No, but it is a *Non-deterministic Finite Automaton*

Deterministic Finite Automata (DFA)

- A DFA is defined by the 5-tuple:
  - {Q, ∑ , q_0,F, Δ }
- Where:
  - Q ==> a finite set of states
  - ∑ ==> a finite set of input symbols (alphabet)
  - q_0 ==> a start state
  - F ==> set of accepting states
  - Δ ==> a transition function, which is a mapping between Q x ∑ ==> Q

How DFA Works

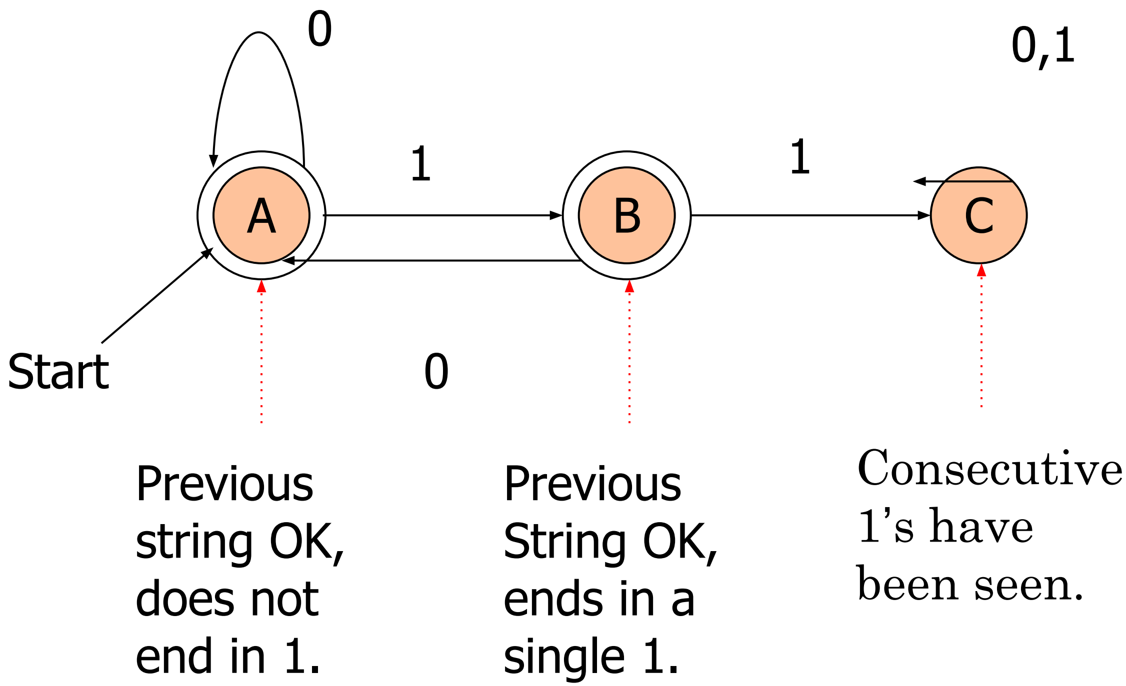
- Input: a word w in $\sum^*$
- Question: Is w acceptable by the DFA?
- Steps:
  - Start at the "start state" q_0
  - For every input symbol in the sequence w do
    - Compute the next state from the current state, given the current input symbol in w and the transition function
  - If after all symbols in w are consumed, the current state is one of the accepting states (F) then *accept w*;
  - Otherwise, *reject w.*

Graph Representation of DFA's

- Nodes = states.
- Arcs represent transition function.
  - Arc from state p to state q labeled by all those input symbols that have transitions from p to q.
- Arrow labeled "Start" to the start state.
- Final states indicated by double circles.

Example: Graph of a DFA

Accepts all strings without two consecutive 1's.



Alternative Representation: Transition Table

Final states starred

Columns = input symbols

Arrow for start state

		0	1
→ *	A	A	B
*	B	A	C
	C	C	C

Rows = states

Extended Transition Function

- We describe the effect of a string of inputs on a DFA by extending δ to a state and a string.
- Induction on length of string.
- Basis: $\delta(q, \varepsilon) = q$
- Induction: $\delta(q,wa) = \delta(\delta(q,w),a)$
  - w is a string; a is an input symbol.

Extended Δ: Intuition

- Convention:
  - … w, x, y, x are strings.
  - a, b, c,… are single symbols.
- Extended δ is computed for state q and inputs $a_1a_2…a_n$ by following a path in the transition graph, starting at q and selecting the arcs with labels a_1, a_2,…,a_n in turn.

Example: Extended Delta

	0	1
A	A	B
B	A	C
C	C	C

$\delta(B,011) = \delta(\delta(B,01),1) = \delta(\delta(\delta(B,0),1),1) =$

$\delta(\delta(A,1),1) = \delta(B,1) = C$

Delta-hat

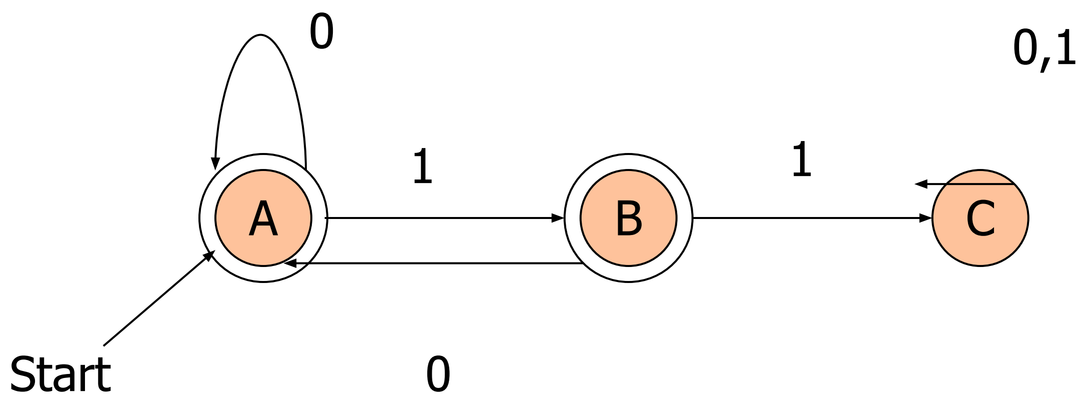
- Some people denote the extended δ with a "hat" to distinguish it from δ itself.
- Not needed, because both agree when the string is a single symbol.
- $\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \epsilon), a) = \delta(q, a)$

Language of a DFA

- Automata of all kinds define languages.
- If A is an automaton, L(A) is its language.
- For a DFA A, L(A) is the set of strings labeling paths from the start state to a final state.
- Formally: L(A) = the set of strings w such that $\delta(q_0, w)$ is in F.

Example: String in a Language

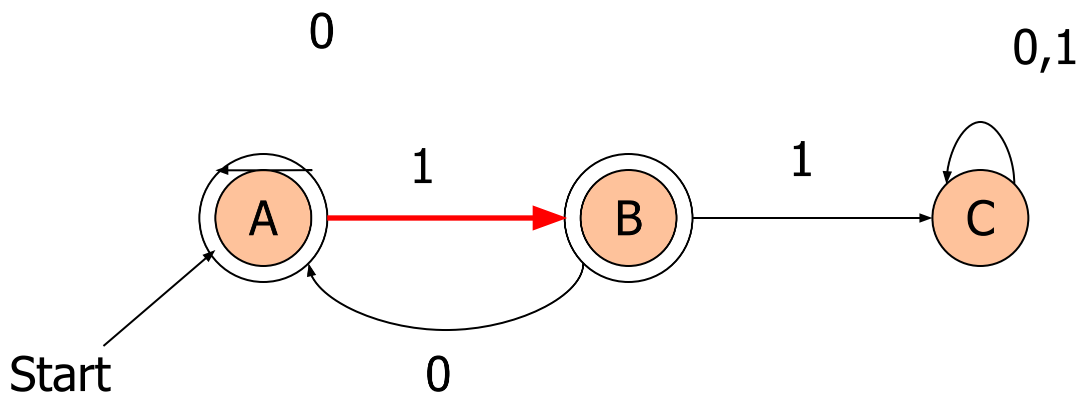
String 101 is in the language of the DFA below.
Start at A.



Example: String in a Language

String 101 is in the language of the DFA below.

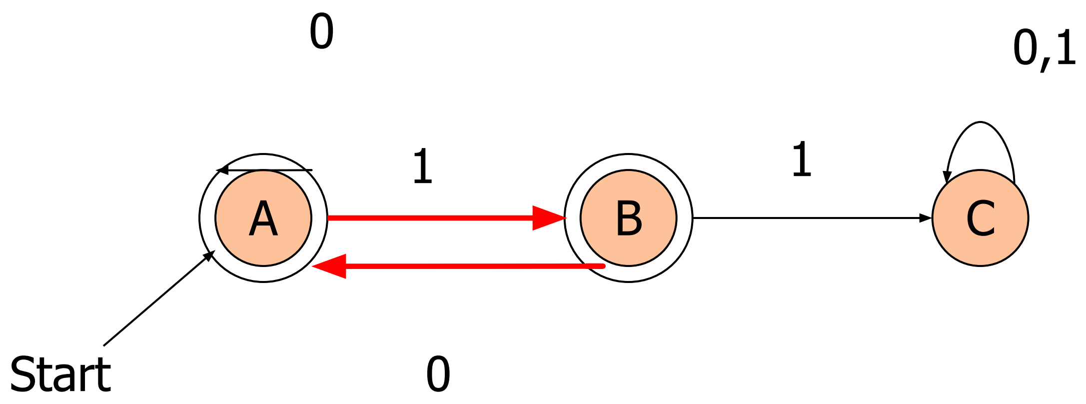
Follow arc labeled 1.



Example: String in a Language

String 101 is in the language of the DFA below.

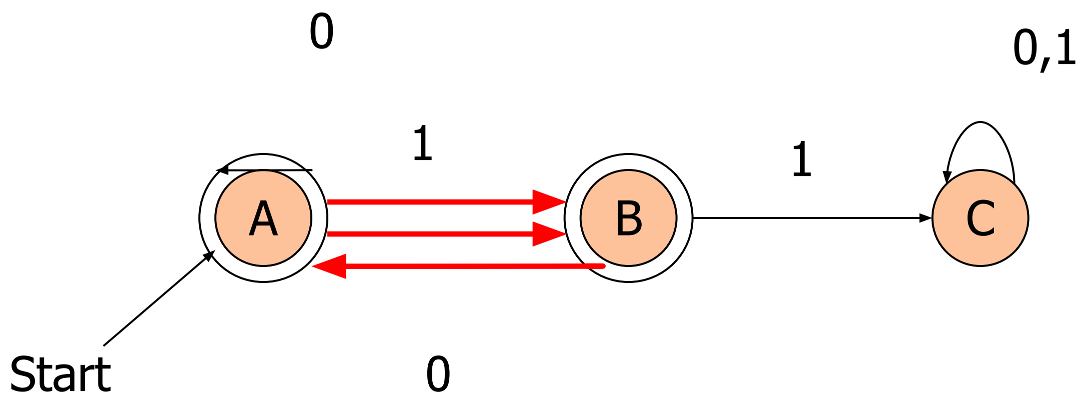
Then arc labeled 0 from current state B.



Example: String in a Language

String 101 is in the language of the DFA below.

Finally arc labeled 1 from current state A. Result is an accepting state, so 101 is in the language.



Example – Concluded

- The language of our example DFA is:

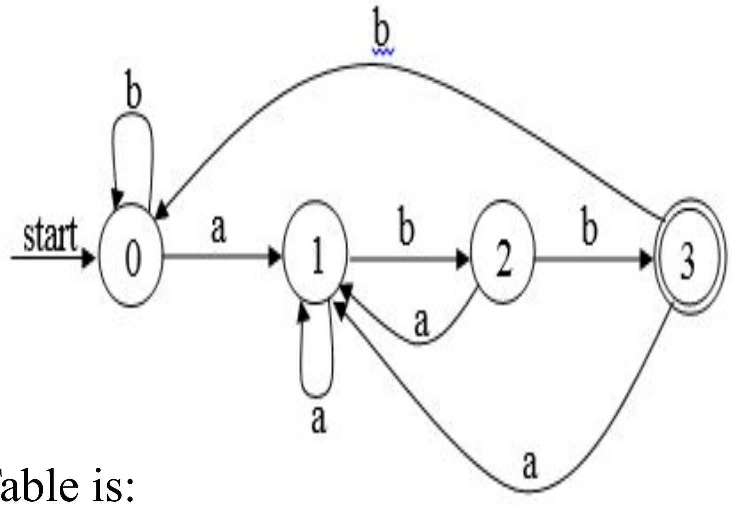
{w | w is in {0,1}* and w does not have two consecutive 1's}

Such that…

These conditions about w are true.

Read a *set former* as "The set of strings w…

Example: The following figure shows a DFA that recognizes the language
(a|b)*abb



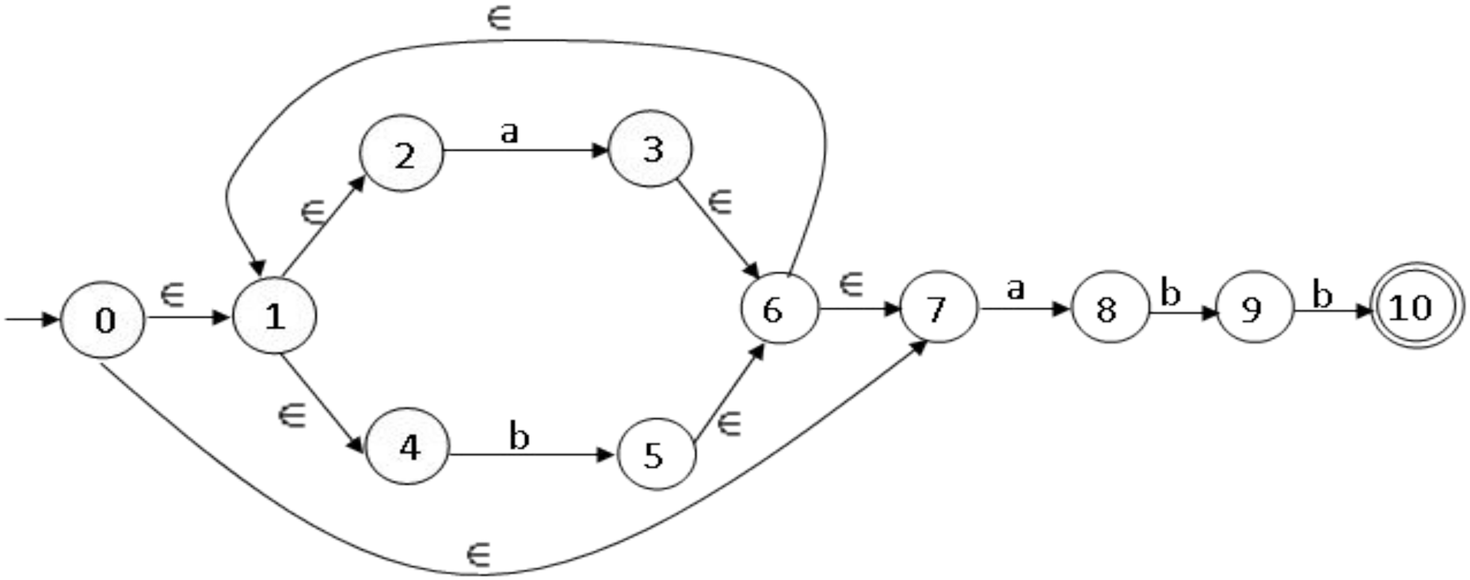
The Transition Table is:

State	a	b
0	1	0
1	1	2
2	1	3
3	1	0

Conversion of an NFA into a DFA

It is hard for a computer program to simulate an NFA because the transition function is multivalued. The algorithm that called the subset construction will convert an NFA for any language into a DFA that recognizes the same languages.

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



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

Sol: apply the Algorithm of Subset construction as follow:

1) Find the start state of the equivalent DFA is ε -closure (0), which is consist of start state of NFA and the all states reachable from state 0 via a path in which every edge is labeled .

A= {0, 1, 2, 4, 7}

2)Compute **move (A, *a*)**, the set of states of **NFA** having transitions on ***a*** from members of **A**. Among the states **0**, **1**, **2**, **4** and **7**, only **2** and **7** have such transitions, to **3** and **8,** so

move (A, *a*)={3, 8}

Compute the -closure (move (A, a)) = ε -closure ({3, 8}),

-closure ({3, 8}) = **{1, 2, 3, 4, 6, 7, 8}** Let us call this set **B**.

- 3) Compute move (A, b), the set of states of NFA having transitions on b from members of A. Among the states 0, 1, 2, 4 and 7, only 4 have such transitions, to 5 so **move (A, *b*)={5}**

Compute the -closure (move (A, b)) = ε-closure ({5}),

ε -closure ({5}) = {1, 2, 4, 5, 6, 7} Let us call this set C.

So the DFA has a transition on b from A to C.

4) We apply the **steps** 2 and 3 on the B and C, this process continues for every new state of the **DFA** until all sets that are states of the **DFA** are marked.

The five different sets of states we actually construct are:

$A = \{0, 1, 2, 4, 7\}$

$B = \{1, 2, 3, 4, 6, 7, 8\}$

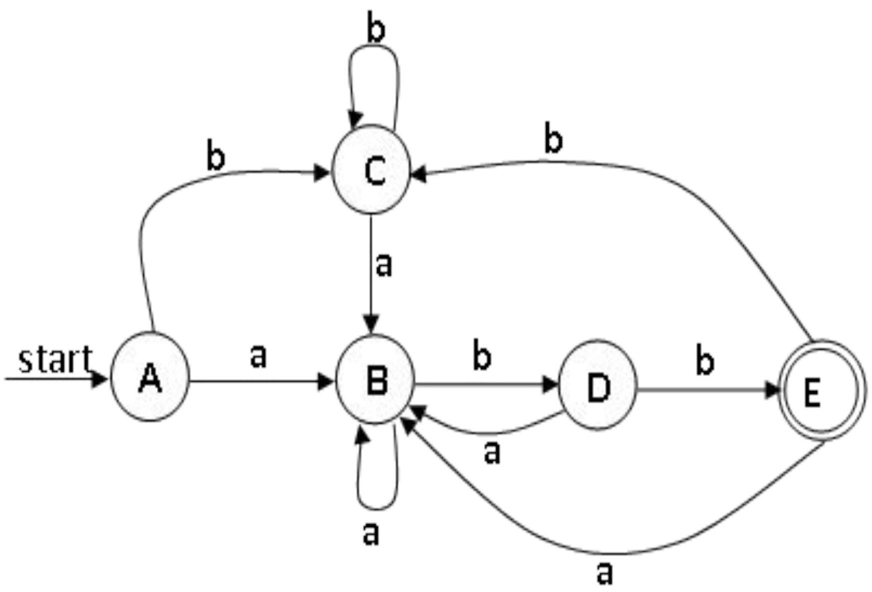
$C = \{1, 2, 4, 5, 6, 7\}$

$D = \{1, 2, 4, 5, 6, 7, 9\}$

$E = \{1, 2, 4, 5, 6, 7, 10\}$

State A is the start state, and state E is the only accepting state. The complete transition table Dtran is shown in below:

STATE	INPUT SYMBOL	
	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



From a Regular Expression to an NFA

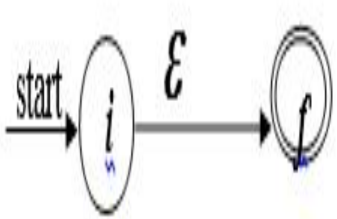
Now give an algorithm to construct an NFA from a regular expression. The algorithm is syntax-directed in that it uses the syntactic structure of the regular expression to guide the construction process.

Algorithm: (Thompson's construction):

Input: a regular expression R over alphabet ∑.

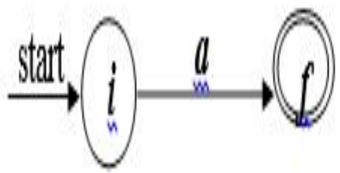
Output: NFA N accepting L(R).

1- For ϵ , construct the NFA

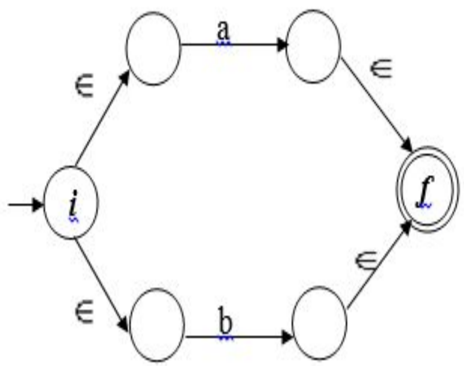


From a Regular Expression to an NFA

2- For ***a*** in ∑, construct the ***NFA***

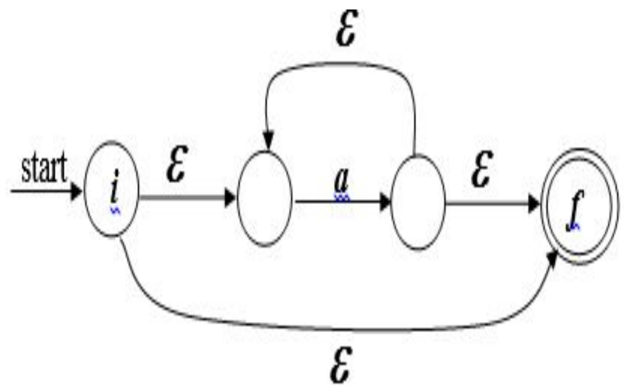


3- For the regular expression ***a | b*** construct the following composite NFA ***N(a | b)***.

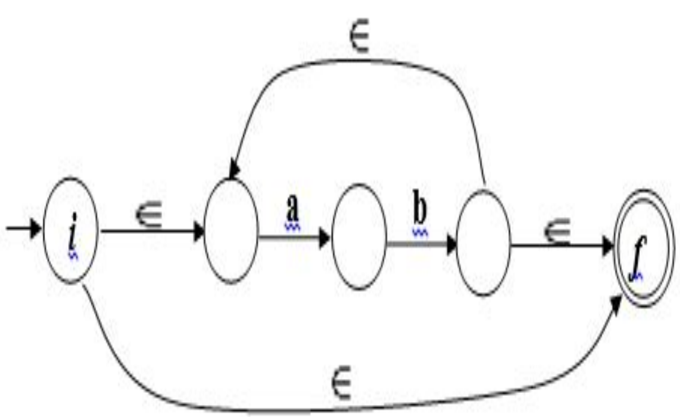


From a Regular Expression to an NFA

4-For the regular expression ***a**** construct the following composite NFA

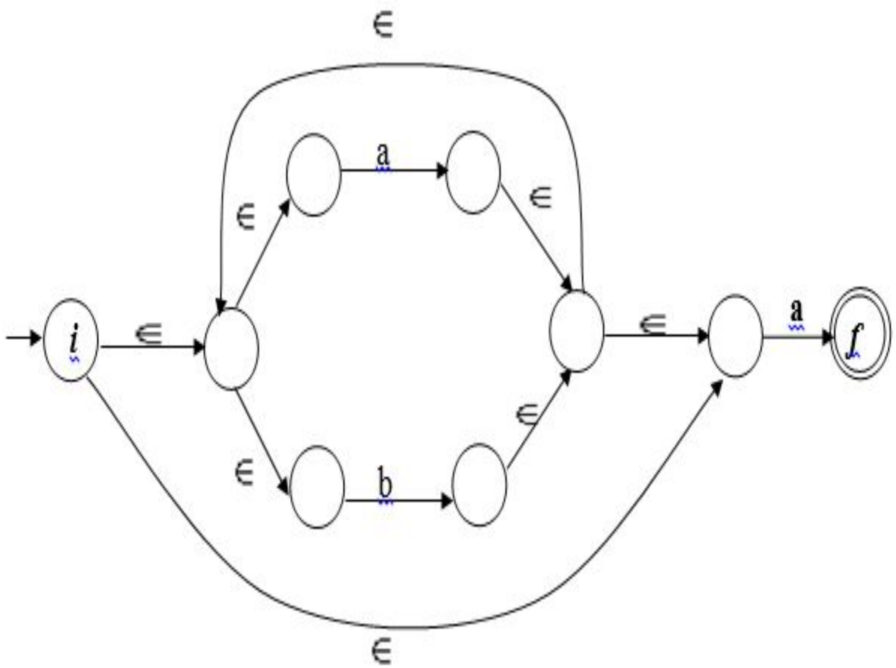


- RE = (***ab***)*



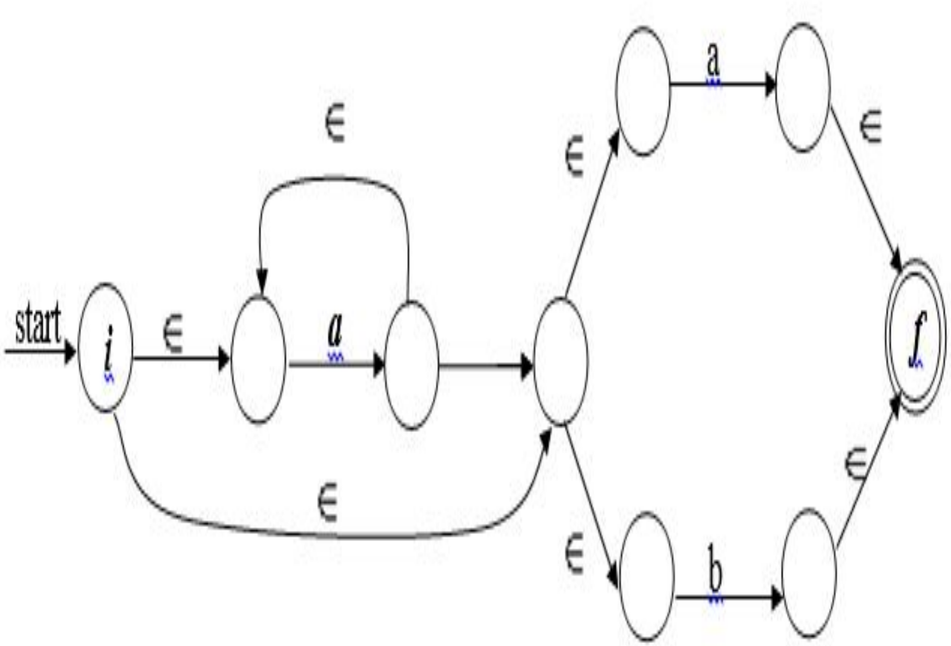
From a Regular Expression to an NFA

RE = $(\boldsymbol{a} \mid \boldsymbol{b})^*\boldsymbol{a}$



From a Regular Expression to an NFA

RE = $\boldsymbol{a}^*\ (\boldsymbol{a} \mid \boldsymbol{b})$



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.

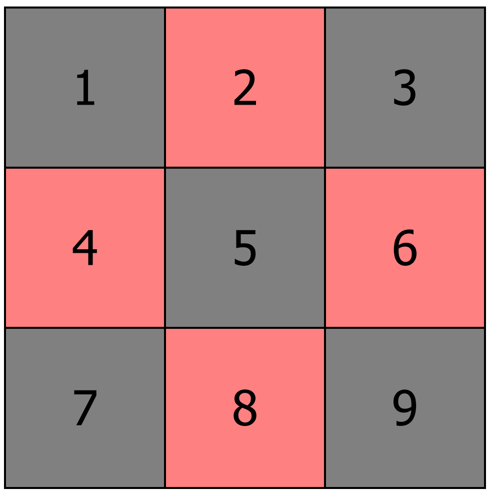
Nondeterminism – (2)

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

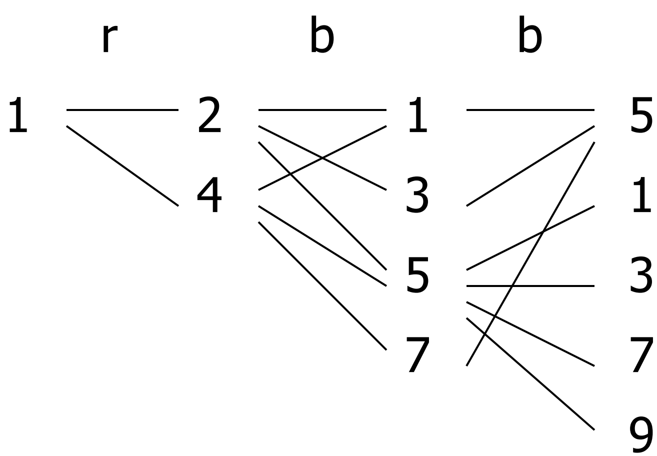
Example: Moves on a Chessboard

- 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

Formal NFA

- A finite set of states, typically Q.
- An input alphabet, typically Σ.
- A transition function, typically δ.
- A start state in Q, typically q_0.
- A set of final states $F \subseteq Q$.

Transition Function of an NFA

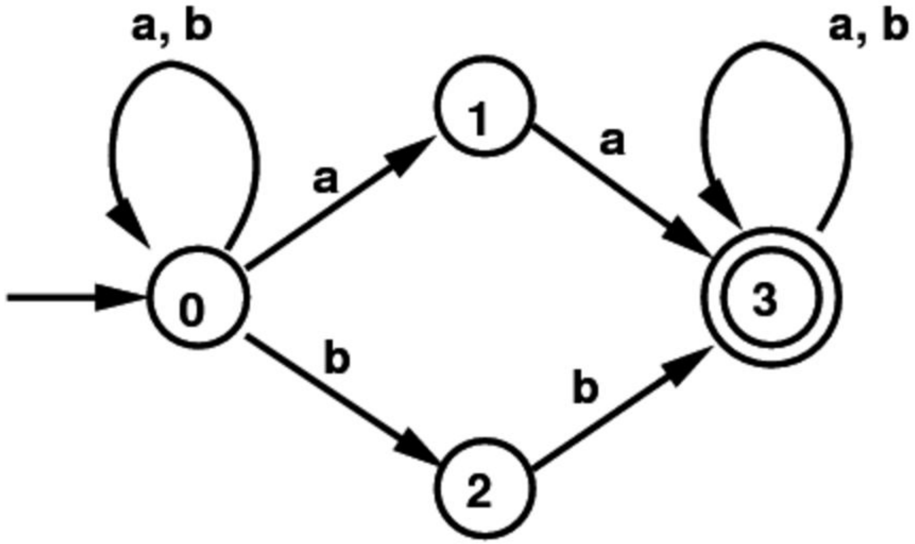
- $\delta(q, a)$ is a set of states.
- Extend to strings as follows:
- Basis: $\delta(q, \epsilon) = \{q\}$
- Induction: $\delta(q, wa)$ = the union over all states p in $\delta(q, w)$ of $\delta(p, a)$

Language of an NFA

- A string w is **accepted** by an NFA if δ(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.

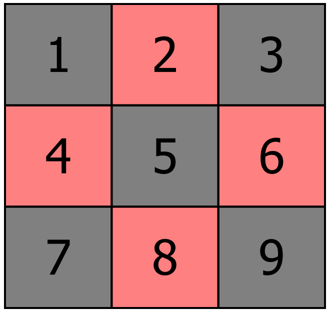
Example NFA

- 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}		

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}

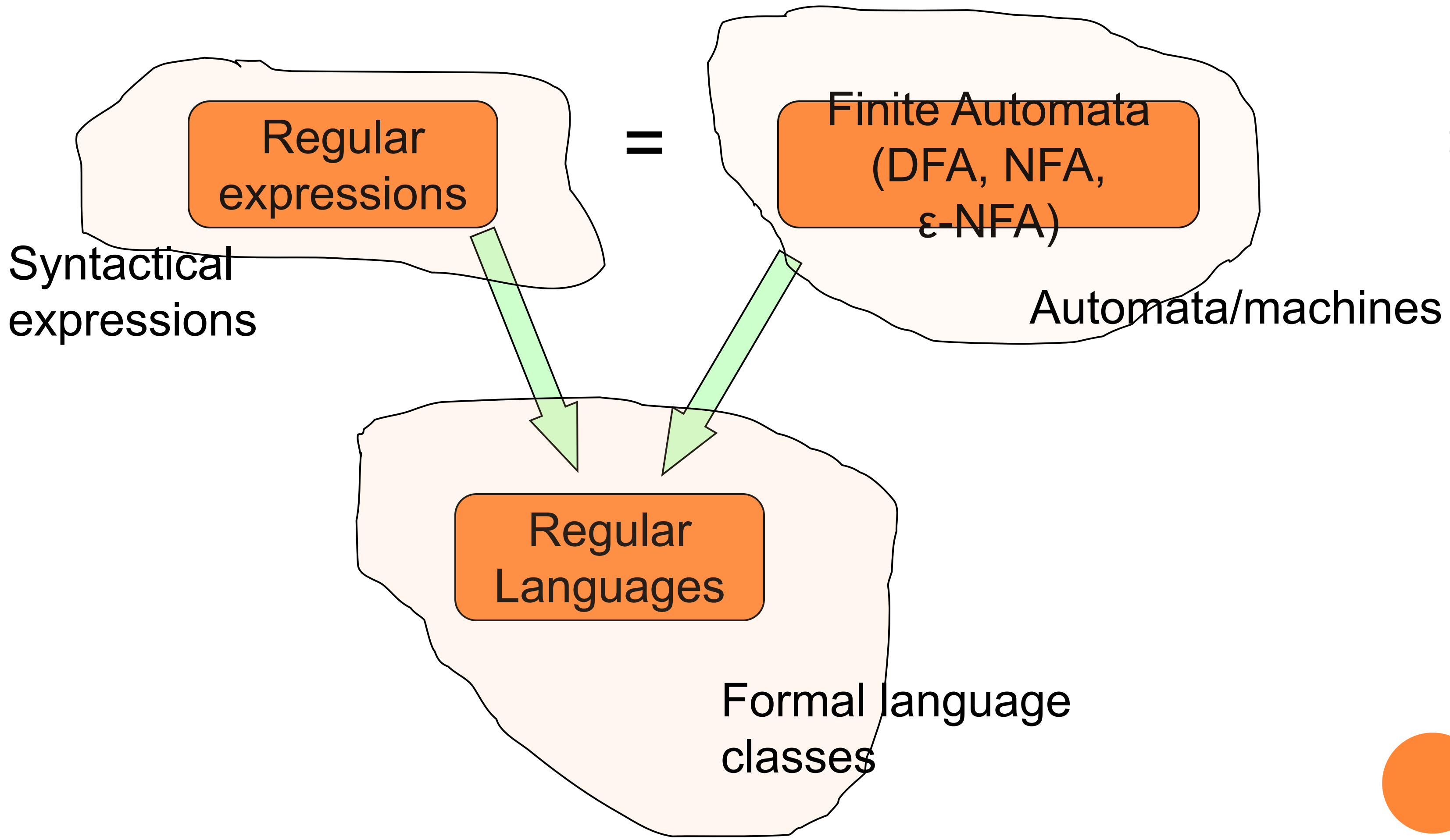
Regular Expressions vs. Finite Automata

- Offers a declarative way to express the pattern of any string we want to accept
  - E.g., 01*+ 10*

- Automata => more machine-like
  < input: string , output: [accept/reject] >
- Regular expressions => more program syntax-like

- Unix environments heavily use regular expressions
  - E.g., bash shell, grep, vi & other editors, sed
- Perl scripting – good for string processing
- Lexical analyzers such as Lex or Flex

Regular Expressions



Language Operators

- Union of two languages:
  - **L U M** = all strings that are either in L or M
  - Note: A union of two languages produces a third language

- Concatenation of two languages:
  - **L . M** = all strings that are of the form *xy* s.t., $x \in L$ and $y \in M$
  - The *dot* operator is usually omitted
    - i.e., **LM** is same as L.M

Kleene Closure (the * operator)

- Kleene Closure of a given language L:
  - $L^0 = \{\varepsilon\}$
  - $L^1 = \{w \mid \text{for some } w \in L\}$
  - $L^2 = \{ w_1w_2 \mid w_1 \in L, w_2 \in L \text{ (duplicates allowed)}\}$
  - $L^i = \{ w_1w_2\ldots w_i \mid \text{all } w\text{'s chosen are} \in L \text{ (duplicates allowed)}\}$
  - (Note: the choice of each w_i is independent)
  - $L^* = \bigcup_{i\geq 0} L^i$ (arbitrary number of concatenations)

Example:

- Let L = { 1, 00}
  - $L^0 = \{\varepsilon\}$
  - $L^1 = \{1,00\}$
  - $L^2 = \{11,100,001,0000\}$
  - $L^3 = \{111,1100,1001,10000,000000,00001,00100,0011\}$
  - $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$

Kleene Closure (special notes)

- L^* is an infinite set iff $|L| \geq 1$ and $L \neq \{\varepsilon\}$ **Why?**
- If $L=\{\varepsilon\}$, then $L^* = \{\varepsilon\}$ **Why?**
- If $L = \Phi$, then $L^* = \{\varepsilon\}$ **Why?**

Σ^* denotes the set of all words over an alphabet Σ

- Therefore, an abbreviated way of saying there is an arbitrary language L over an alphabet Σ is:
  - $L \subseteq \Sigma^*$

Building Regular Expressions

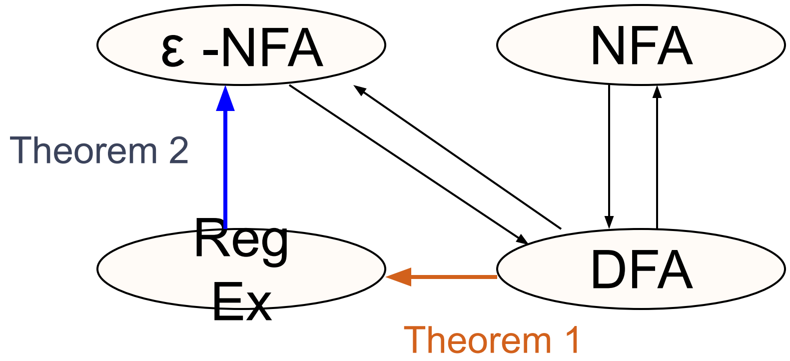
- Let E be a regular expression and the language represented by E is L(E)
- Then:
  - (E) = E
  - L(E + F) = L(E) U L(F)
  - L(E F) = L(E) L(F)
  - L(E*) = (L(E))*

Precedence of Operators

- Highest to lowest
  - * operator (star)
  - .   (concatenation)
  - + operator
- Example:
  - $01^* + 1 \quad = \quad (\, 0 \,.\, ((1)^*) \,) + \; 1$

Finite Automata (FA) & Regular Expressions (Reg Ex)

- To show that they are interchangeable, consider the following theorems:
  - *Theorem 1:* *For every DFA A there exists a regular expression R such that L(R)=L(A)*
  - *Theorem 2:* *For every regular expression R there exists an ε -NFA E such that L(E)=L(R)*

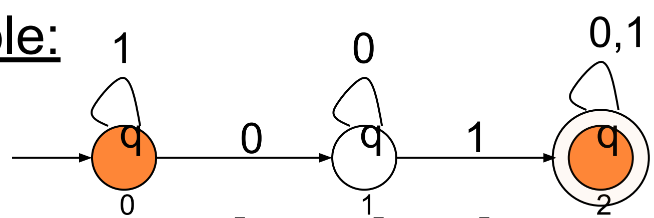


Kleene Theorem

DFA to RE construction

Informally, trace all distinct paths (traversing cycles only once) from the start state to *each of the* final states and enumerate all the expressions along the way

<u>Example:</u>



$$\underbrace{(1^*)}_{1^*}\ \underbrace{0\ (0^*)}_{00^*}\ \underbrace{1}_{1}\ \underbrace{(0+1)^*}_{(0+1)^*}$$

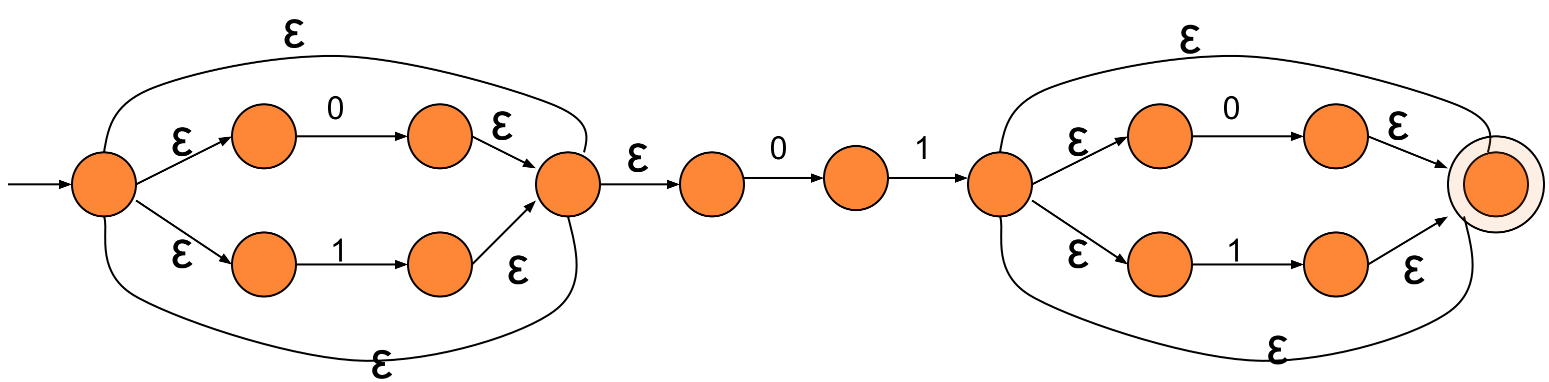
$$1^*00^*1(0+1)^*$$

Q) What is the language?

RE to e-NFA construction

Example: (0+1)*01(0+1)*

(0+1)* 01 (0+1)*



Algebraic Laws of Regular Expressions

- Commutative:
  - $E+F = F+E$
- Associative:
  - $(E+F)+G = E+(F+G)$
  - $(EF)G = E(FG)$
- Identity:
  - $E+\Phi = E$
  - $\varepsilon E = E \varepsilon = E$
- Annihilator:
  - $\Phi E = E\Phi = \Phi$

Algebraic Laws…

- Distributive:
  - $E(F+G) = EF + EG$
  - $(F+G)E = FE+GE$
- Idempotent: $E + E = E$
- Involving Kleene closures:
  - $(E^*)^* = E^*$
  - $\Phi^* = \varepsilon$
  - $\varepsilon^* = \varepsilon$
  - $E^+ = EE^*$
  - $E? = \varepsilon + E$

True or False?

Let R and S be two regular expressions. Then:

1. $((R^*)^*)^* = R^*$?
2. $(R+S)^* = R^* + S^*$?
3. $(RS + R)^* RS = (RR^*S)^*$?

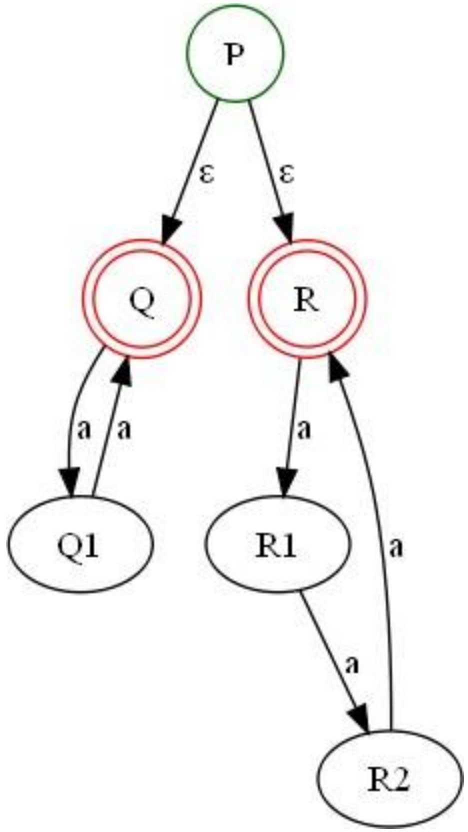
NFA's with ε –Transitions

- We extend the class of NFAs by allowing instantaneous (ε) transitions:

1. The automaton may be allowed to change its state without reading the input symbol.
2. In diagrams, such transitions are depicted by labeling the appropriate arcs with ε.
3. Note that this does not mean that ε has become an input symbol. On the contrary, we assume that *the symbol* ε *does not belong to any alphabet*.

EXAMPLE

- { a^n | n is even or divisible by 3 }



DEFINITION

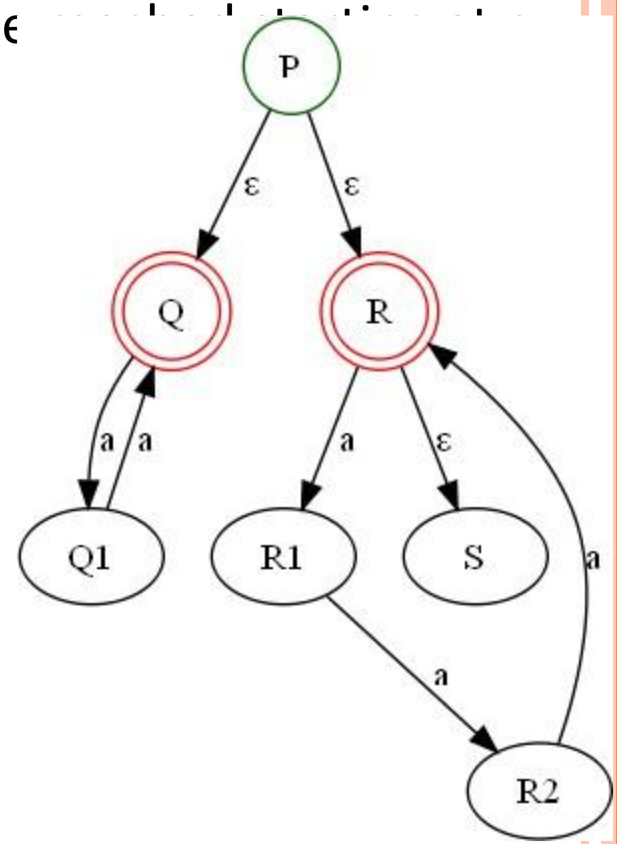
- **A** ε-NFA is a quintuple $\mathbf{A=(Q,\Sigma,\delta,q_0,F)}$ where
  - $\mathbf{Q}$ is a set of *states*
  - $\mathbf{\Sigma}$ is the alphabet of *input symbols*
  - $\mathbf{q_0 \in Q}$ is the *initial state*
  - $\mathbf{F} \subseteq \mathbf{Q}$ is the set of *final states*
  - $\boldsymbol{\delta}: \mathbf{Q} \times \Sigma_\varepsilon \longrightarrow P(\mathbf{Q})$ is the *transition function*
- Note ε is never a member of Σ
- Σ_ε is defined to be $(\Sigma \cup \varepsilon)$

ε -NFA

- ε -NFAs add a convenient feature but (in a sense) they bring us nothing new: they do not extend the class of languages that can be represented. Both NFAs and ε-NFAs recognize exactly the same languages.
- ε-transitions are a convenient feature: try to design an NFA for the even or divisible by 3 language that does not use them!
  - Hint, you need to use something like the product construction from union-closure of DFAs

ε -Closure

- ε-closure of a state
- The ε-*closure* of the state q, denoted ECLOSE(q), is the set that contains q, together with all states that can be



by following only ε-transitions.

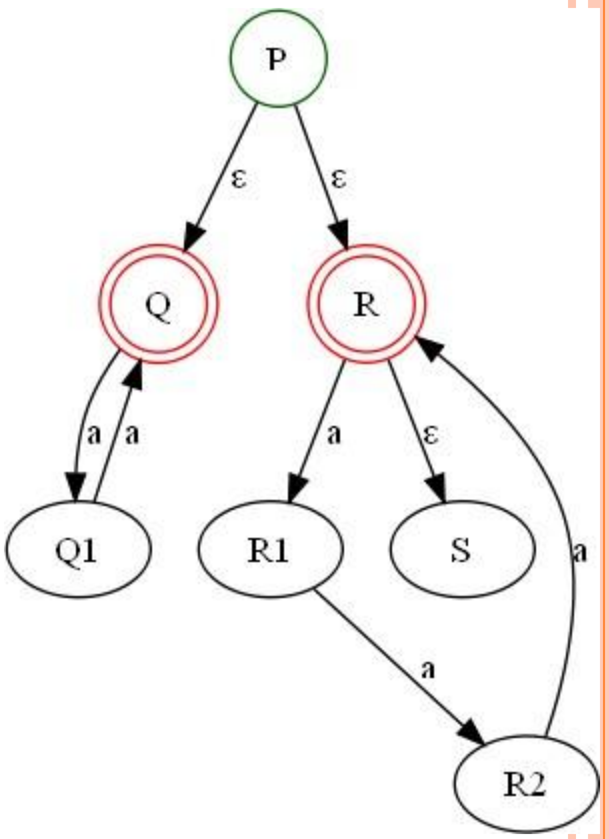
- In the above example:
- ECLOSE(P) ={P,Q,R,S}
- ECLOSE(R)={R,S}
- ECLOSE(x)={x} for the remaining 5 states

{Q,Q1,R1,R2,R2}

Computing eclose

- Compute eclose by adding new states until no new states can be added

- Start with [P]
- Add Q and R to get [P,Q,R]
- Add S to get [P,Q,R S]
- No new states can be added



Elimination of ε -Transitions

- Given an ε-NFA N, this construction produces an NFA N' such that L(N')=L(N).
- The construction of N' begins with N as input, and takes 3 steps:

1. Make p an accepting state of N' iff ECLOSE(p) contains an accepting state of N.
2. Add an arc from p to q labeled a iff there is an arc labeled a in N from some state in ECLOSE(p) to q.
3. Delete all arcs labeled ε.

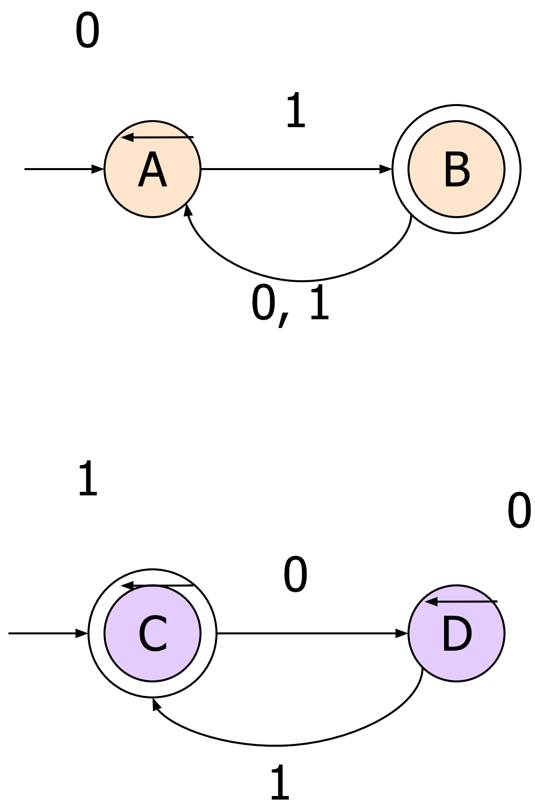
Decision Property: Equivalence

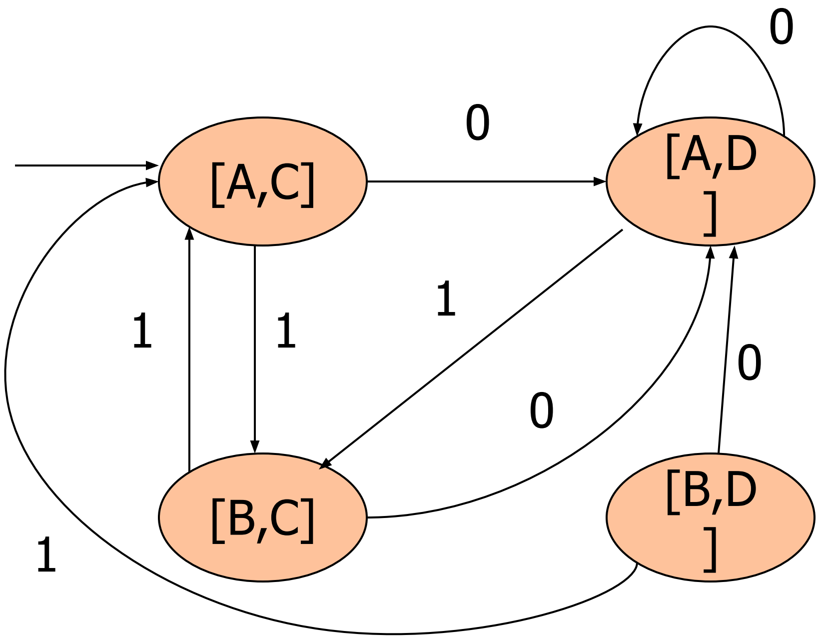
- Given regular languages L and M, is L = M?
- Algorithm involves constructing the *product DFA* from DFA's for L and M.
- Let these DFA's have sets of states Q and R, respectively.
- Product DFA has set of states Q × R.
  - I.e., pairs [q, r] with q in Q, r in R.

Product DFA – Continued

- Start state = $[q_0, r_0]$ (the start states of the DFA's for L, M).
- Transitions: $\delta([q,r], a) = [\delta_L(q,a), \delta_M(r,a)]$
  - δ_L, δ_M are the transition functions for the DFA's of L, M.
  - That is, we simulate the two DFA's in the two state components of the product DFA.

Example: Product DFA

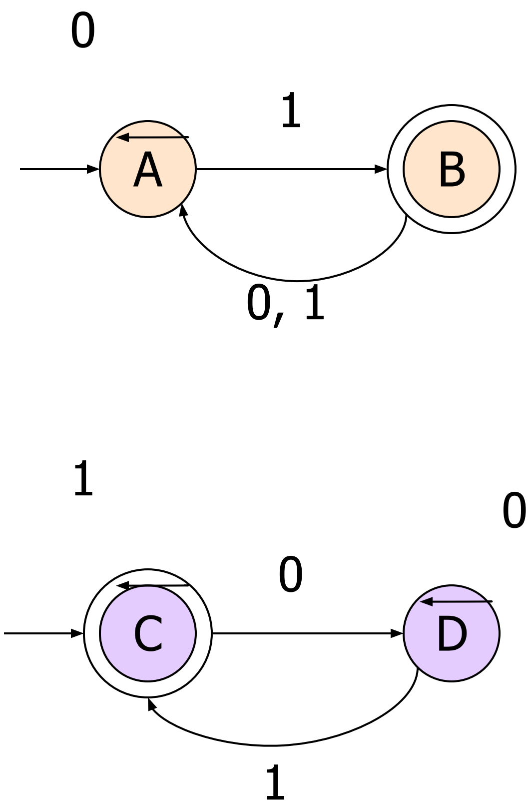


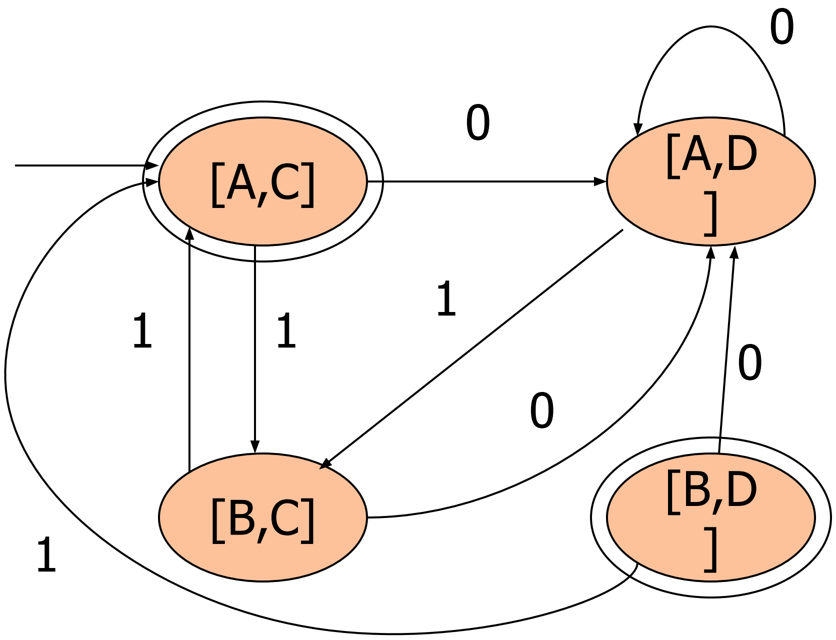


Equivalence Algorithm

- Make the final states of the product DFA be those states [q, r] such that exactly one of q and r is a final state of its own DFA.
- Thus, the product accepts w iff w is in exactly one of L and M.

Example: Equivalence





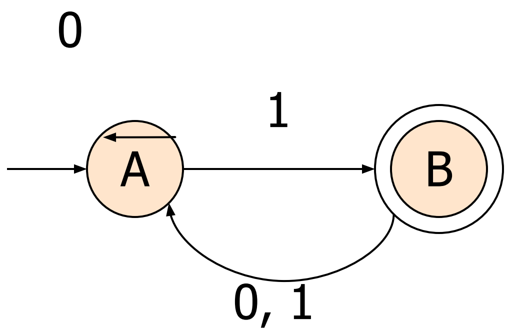
Equivalence Algorithm – (2)

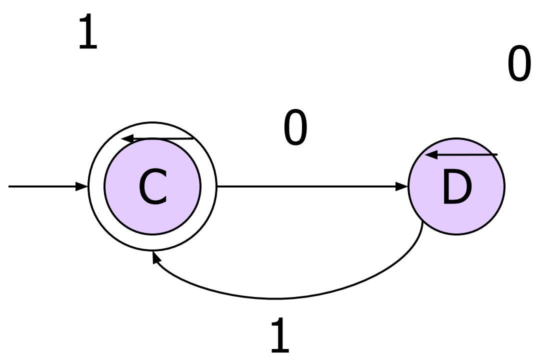
- The product DFA's language is empty iff L = M.
- But we already have an algorithm to test whether the language of a DFA is empty.

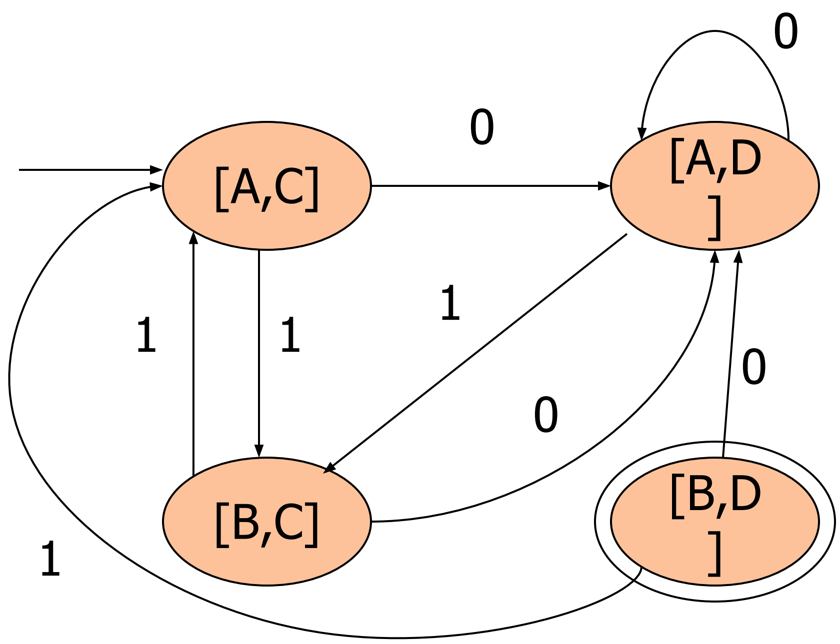
Decision Property: Containment

- Given regular languages L and M, is $L \subseteq M$?
- Algorithm also uses the product automaton.
- How do you define the final states [q, r] of the product so its language is empty iff $L \subseteq M$?

Example: Containment







Note: the only final state is unreachable, so containment holds.

The Minimum-State DFA for a Regular Language

- In principle, since we can test for equivalence of DFA's we can, given a DFA *A* find the DFA with the fewest states accepting L(A).
- Test all smaller DFA's for equivalence with *A*.
- But that's a terrible algorithm.

Efficient State Minimization

- Construct a table with all pairs of states.
- If you find a string that *distinguishes* two states (takes exactly one to an accepting state), mark that pair.
- Algorithm is a recursion on the length of the shortest distinguishing string.

State Minimization – (2)

- Basis: Mark a pair if exactly one is a final state.
- Induction: mark [q, r] if there is some input symbol a such that [δ(q,a), δ(r,a)] is marked.
- After no more marks are possible, the unmarked pairs are equivalent and can be merged into one state.

TRANSITIVITY OF "INDISTINGUISHABLE"

- If state p is indistinguishable from q, and q is indistinguishable from r, then p is indistinguishable from r.
- Proof: The outcome (accept or don't) of p and q on input w is the same, and the outcome of q and r on w is the same, then likewise the outcome of p and r.

Constructing the Minimum-State DFA

- Suppose $q_1,\ldots,q_k$ are indistinguishable states.
- Replace them by one state q.
- Then $\delta(q_1, a),\ldots, \delta(q_k, a)$ are all indistinguishable states.
  - Key point: otherwise, we should have marked at least one more pair.
- Let $\delta(q, a)$ = the representative state for that group.

Example: State Minimization

		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}

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

Here it is with more convenient state names

Remember this DFA? It was constructed for the chessboard NFA by the subset construction.

Example – Continued

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
	F	D	C
*	G	D	G
*			

	G	F	E	D	C	B
A	x	x				
B	x	x				
C	x	x				
D	x	x				
E	x	x				
F	x	x				

Start with marks for the pairs with one of the final states F or G.

Example – Continued

	r	b
→ A	B	C
B	D	E
C	D	F
D	D	G
E	D	G
F	D	C
* G	D	G
*		

	G	F	E	D	C	B
A	x	x				
B	x	x				
C	x	x				
D	x	x				
E	x	x				
F	x	x				

Input r gives no help, because the pair [B, D] is not marked.

Example – Continued

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
	F	D	C
*	G	D	G
*			

	G	F	E	D	C	B
A	x	x	x	x	x	
B	x	x	x	x	x	
C	x	x				
D	x	x				
E	x	x				
F	x					

But input b distinguishes {A,B,F} from {C,D,E,G}. For example, [A, C] gets marked because [C, F] is marked.

Example – Continued

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
	F	D	C
*	G	D	G
*			

	G	F	E	D	C	B
A	x	x	x	x	x	
B	x	x	x	x	x	
C	x	x	x	x		
D	x	x				
E	x	x				
F	x					

[C, D] and [C, E] are marked because of transitions on b to marked pair [F, G].

Example – Continued

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
	F	D	C
*	G	D	G
*			

	G	F	E	D	C	B
A	x	x	x	x	x	x
B	x	x	x	x	x	
C	x	x	x	x		
D	x	x				
E	x	x				
F	x					

[A, B] is marked
because of transitions on r
to marked pair [B, D].

[D, E] can never be marked,
because on both inputs they
go to the same state.

Example – Concluded

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
	F	D	C
*	G	D	G
*			

		r	b
→	A	B	C
	B	H	H
	C	H	F
	H	H	G
	F	H	C
*	G	H	G
*			

	G	F	E	D	C	B
A	x	x	x	x	x	x
B	x	x	x	x	x	
C	x	x	x	x		
D	x	x				
E	x	x				
F	x					

Replace D and E by H.

Result is the minimum-state DFA.

END