Theory of Computer Science – SCJ 3203

Finite Automata

Sazali Abd Manaf
Mohd Soperi Mohd Zahid

cc BY NC SA

ocw.utm.my

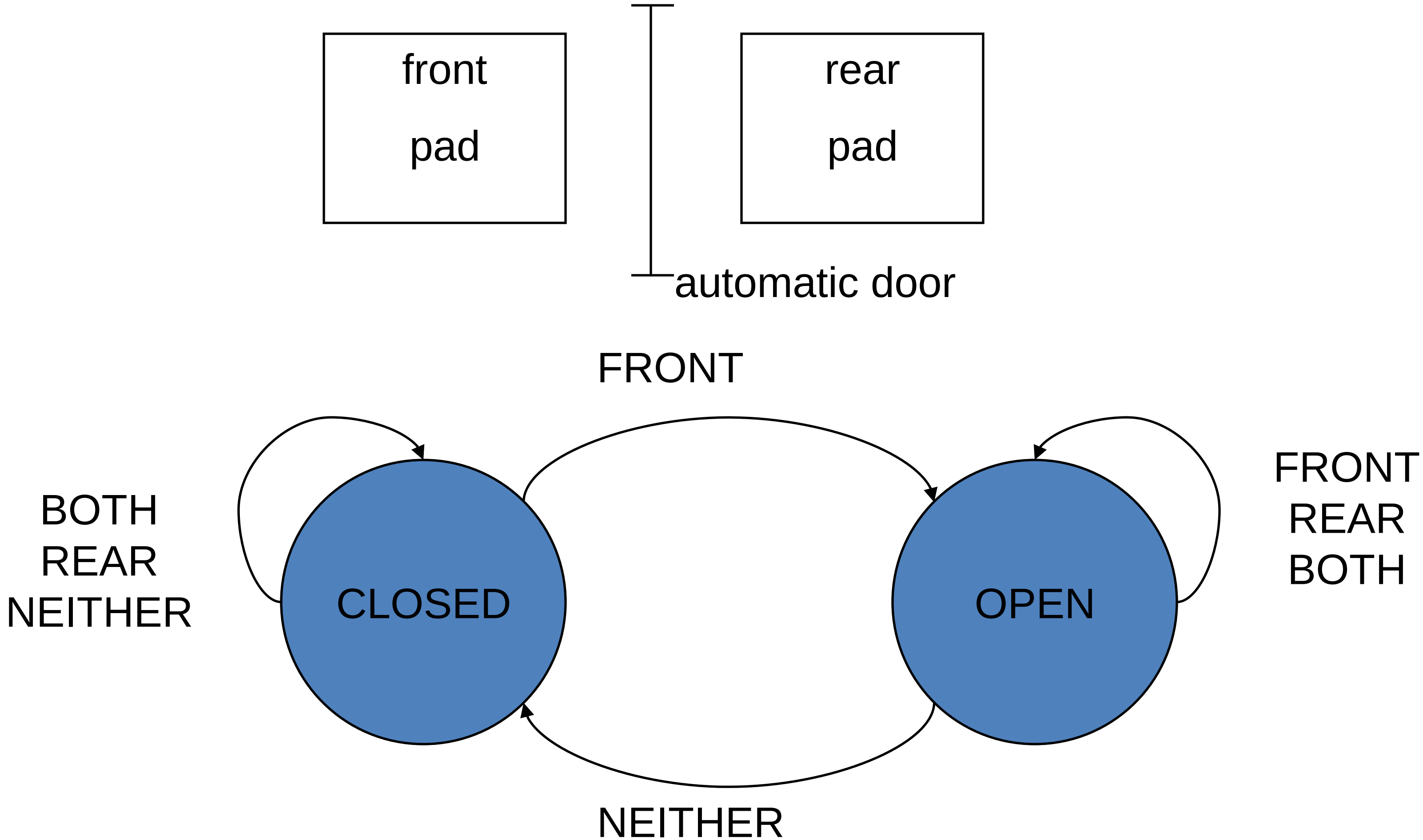
Outline

- Deterministic Finite Automaton (DFA)
- Non-Deterministic Finite Automaton (NFA)
- Regular Expressions and Languages

Finite Automata (FA)

- A simple class of machines with limited capabilities.
- good models for computers with an extremely limited amount of memory.
- e.g., an automatic door : a computer with only a single bit of memory

State Diagram



State Transition Table

		INPUT SIGNAL			
		NEITHER	FRONT	REAR	BOTH
STATE	CLOSED	CLOSED	OPEN	CLOSED	CLOSED
	OPEN	CLOSED	OPEN	OPEN	OPEN

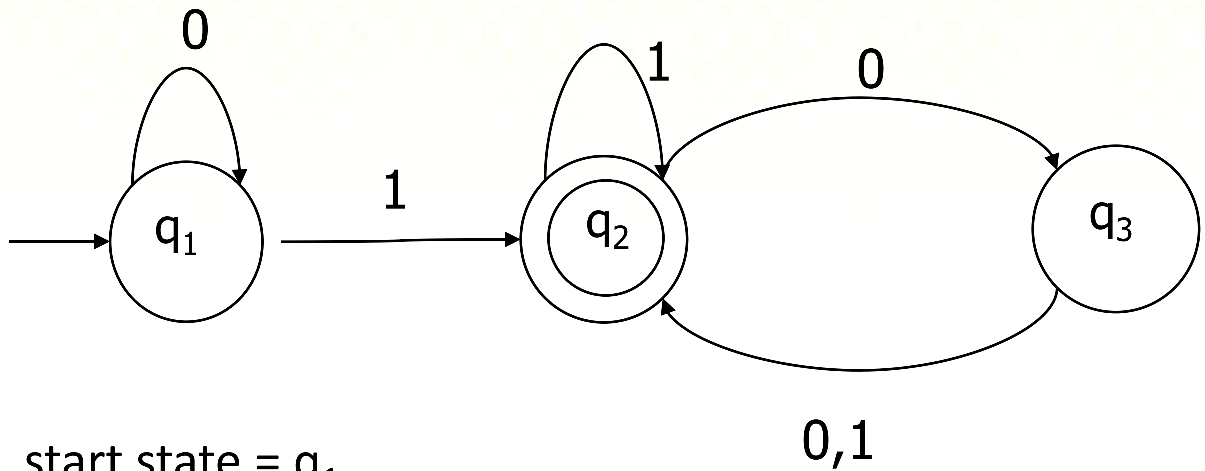
Examples

- Elevator controller
  - state : floor
  - input : signal received from the buttons.
- Dishwashers
- Electronic thermostats
- Digital watches
- Calculators

Definition of fa

- A finite automaton **(fa)** is a collection of 3 things:
  - A finite set of states, one of which is designated as the initial state, called the **start state**, and some (maybe none) of which are designated as **final states**.
  - An **alphabet** Σ of possible input letters.
  - A finite set of **transition rules** that tell for each state and for each letter of the input alphabet which state to go to next.

State Diagram



- start state = q_1
- final state = q_2
- transitions = each arrows
- alphabet = each labels

- When this automaton receives an input string such as 1101, it processes that string and produce output (Accept or Reject).

cc BY NC SA

Language of machine

- If A is the set of all strings that machine M accepts, we say that A is the language of machine M.

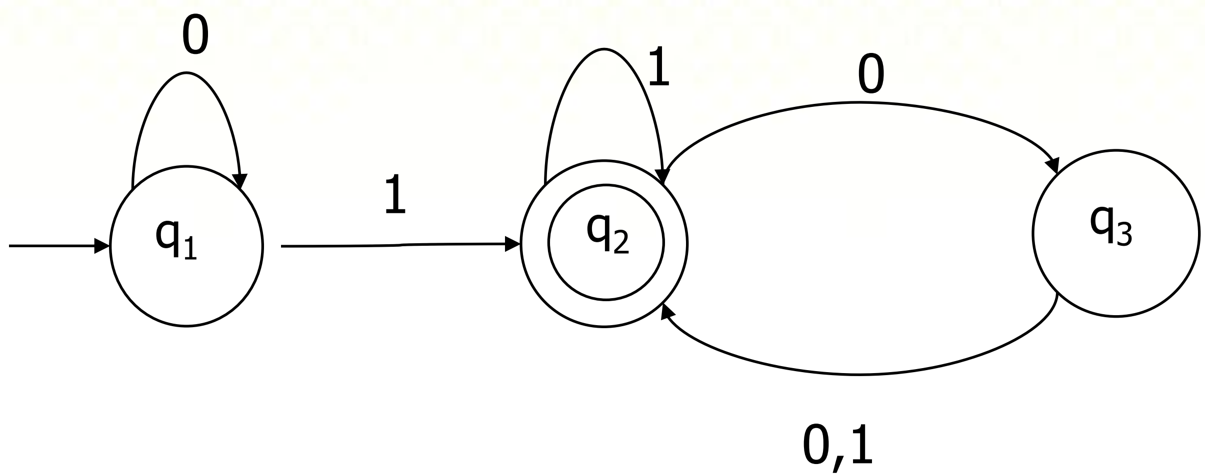
$$L(M) = A$$

- M recognizes A (only 1 language)
- M accepts strings (several strings)
- If M accepts no strings, it still recognizes one language, empty language $\varnothing$

Formal Definition

- A finite automaton is a 5-tuple **(Q,Σ,δ,q_0,F)** where
  - **Q** is a finite set called the **states**,
  - Σ is a finite set called the **alphabet**,
  - δ: $Q \times \Sigma \rightarrow Q$ is the **transition function**,
  - **q_0** $\in Q$ is the **start state**, and
  - **F** $\subseteq Q$ is the **set of accept states (final states)**

Example : Finite Automaton M_1

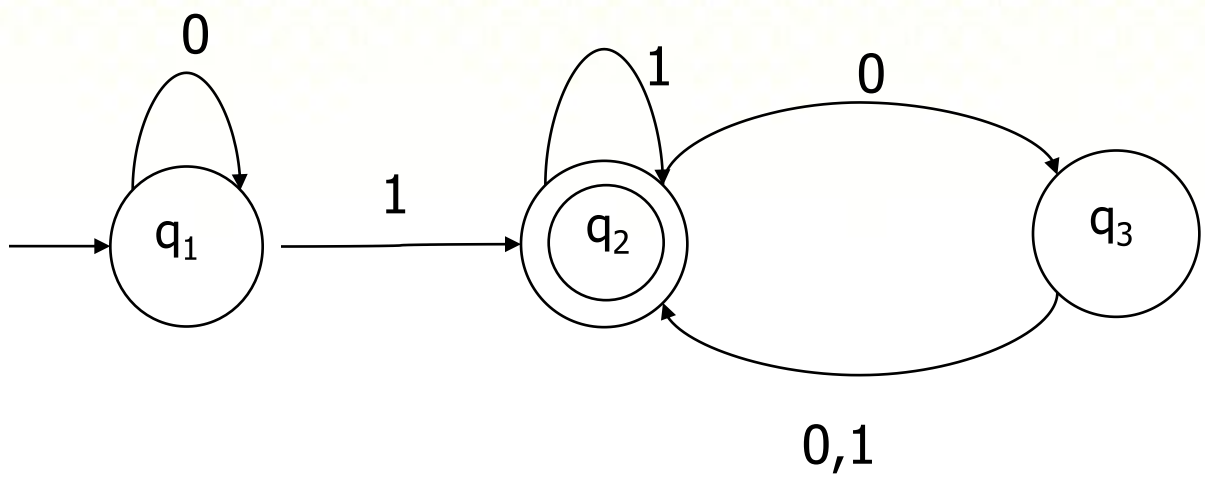


- $M_1 = (Q, \Sigma, \delta, q_0, F)$, where
  - $Q = \{q_1, q_2, q_3\}$,
  - $\Sigma = \{0,1\}$,
  - δ is described as
  - q_1 is the **start state**, and
  - $F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

CC BY NC SA

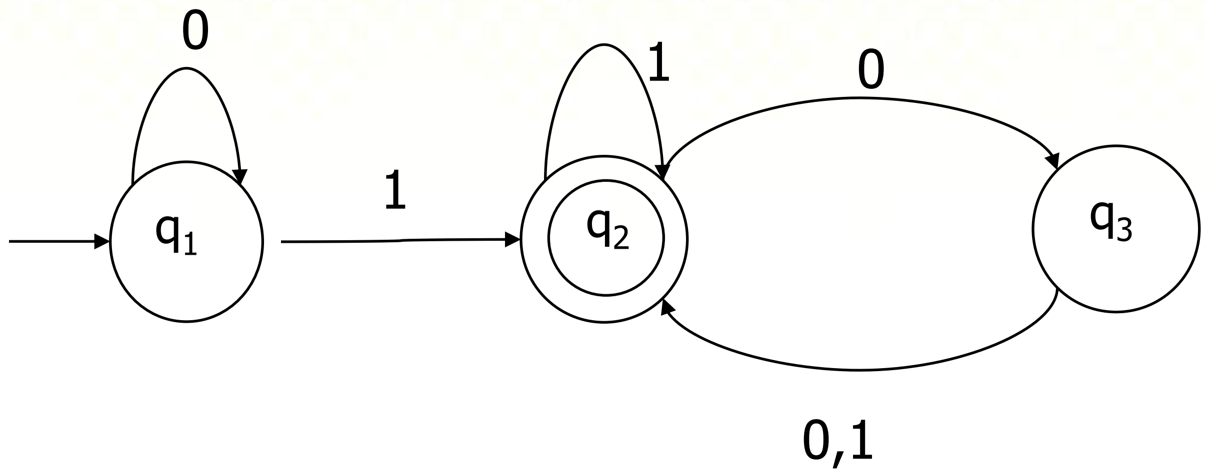
Example : Finite Automaton M_1



What is the language of M_1?

BY NC SA

Example : Finite Automaton M_1

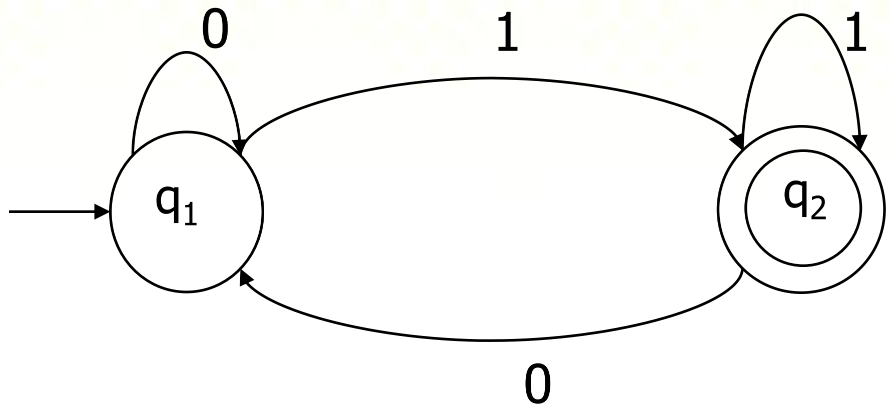


- A = {w | w contains at least one 1 and an even number of 0s follow that last 1}

$L(M_1) = A$, or equivalently, M1 recognizes A

cc BY NC SA

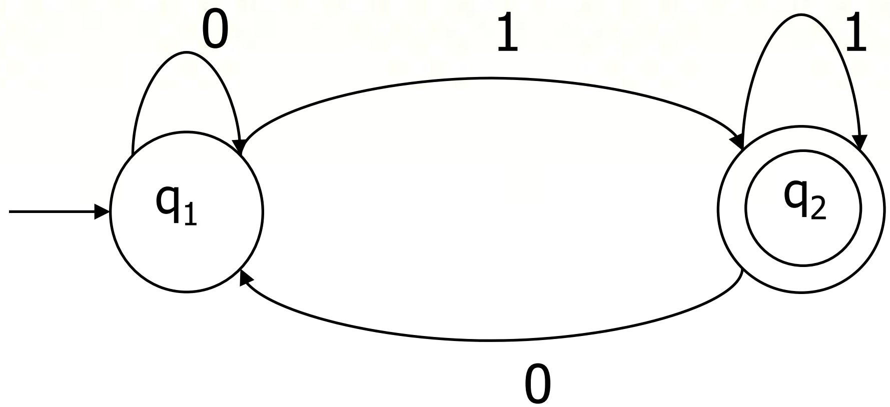
Example : Finite Automaton M_2



- $M_2 = (Q, \Sigma, \delta, q_0, F)$, where
  - Q =
  - Σ =
  - δ is described as
  - is the **start state**, and
  - F = { }.

	0	1
q_1		
q_2		

Example : Finite Automaton M_2

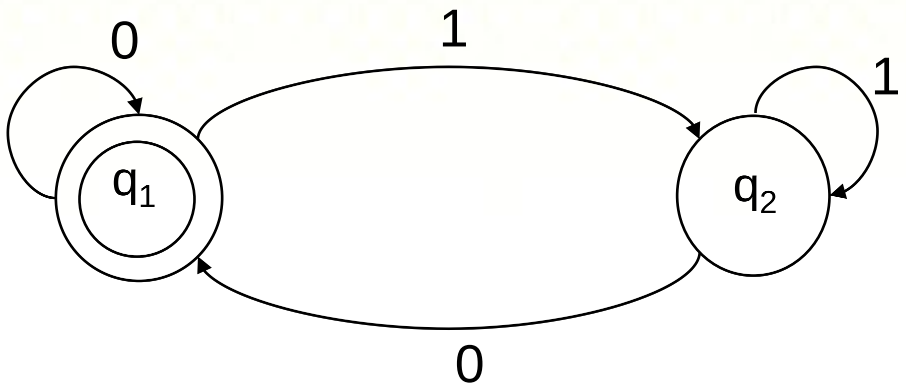


What is the language of M_2?

$L(M_2) = \{w \mid w \text{ ends in a } 1\}$

BY NC SA

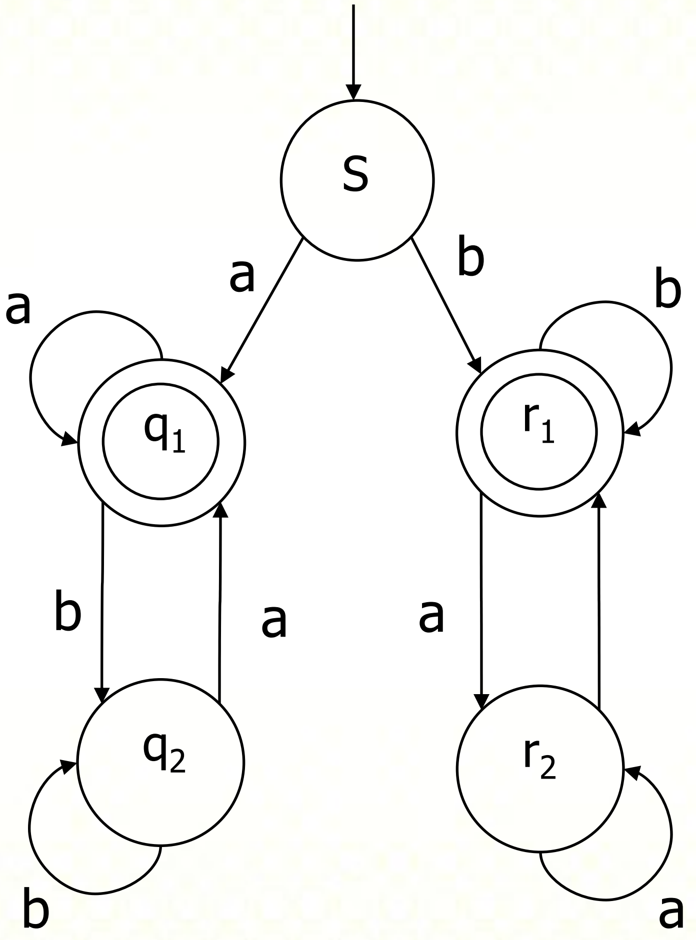
Empty String ε



- If the start state is also a final state, what string does it automatically accept ?
- $L(M_3) = \{ w \mid w \text{ is the empty string } \varepsilon \text{ or ends in a } 0\}$

cc BY NC SA

Example : Finite Automaton M_4



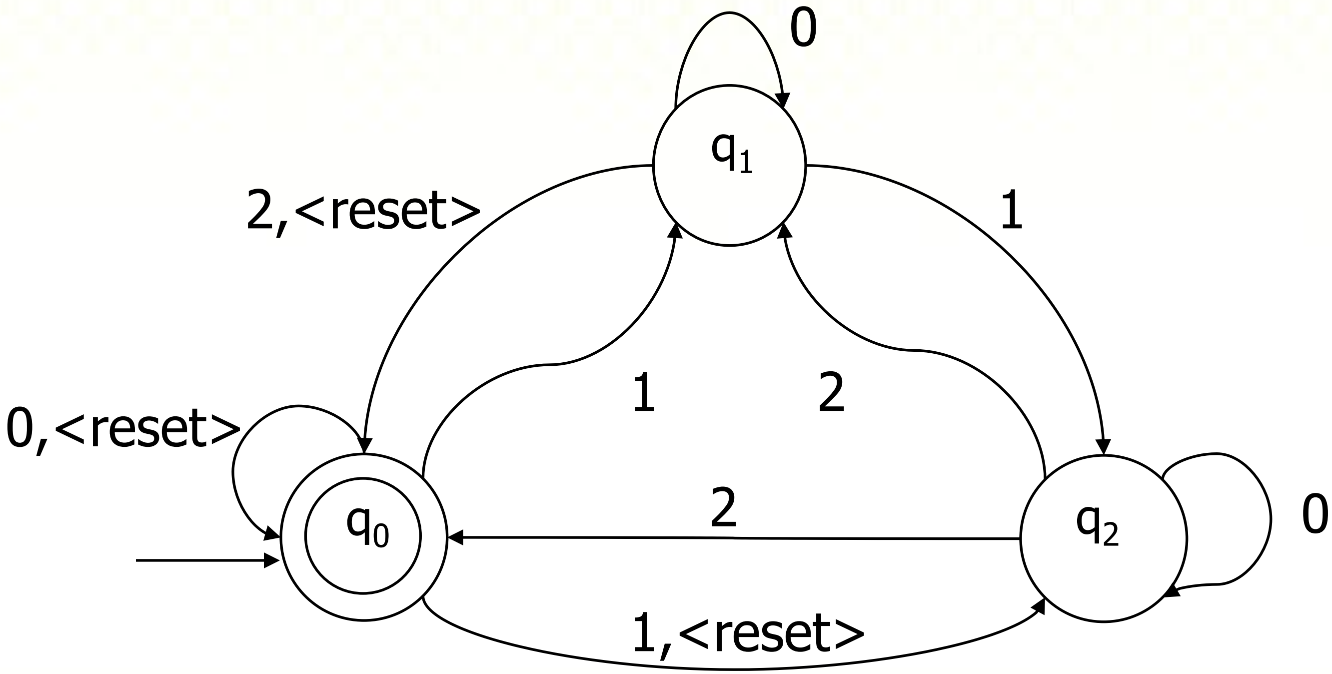
L(M_4) =

- M_4= (Q,Σ,δ,q_0,F) , where
  - Q =
  - Σ =
  - δ is described as

	a	b
q_1		
q_2		
r_1		
r_2		

  - ? is the **start state**, and
  - F = { }.

Example : Finite Automaton M_5



- $\sum$ = {<reset>, 0, 1, 2}
- we treat <reset> as a single symbol.
- What does the M_5 accept ?

cc BY NC SA

FA with Computer Language

- Computer Language
  - Certain character strings are recognizable words. (DO, IF,END,…)
  - Certain strings of words are recognizable commands.
  - Certain set of commands become a program that can be compiled which means translated into machine commands.
- FA is used to determine whether the input commands (instruction) is valid or not corresponding to the structure rules.
- FA implements the rule with the transitions.

Determinism

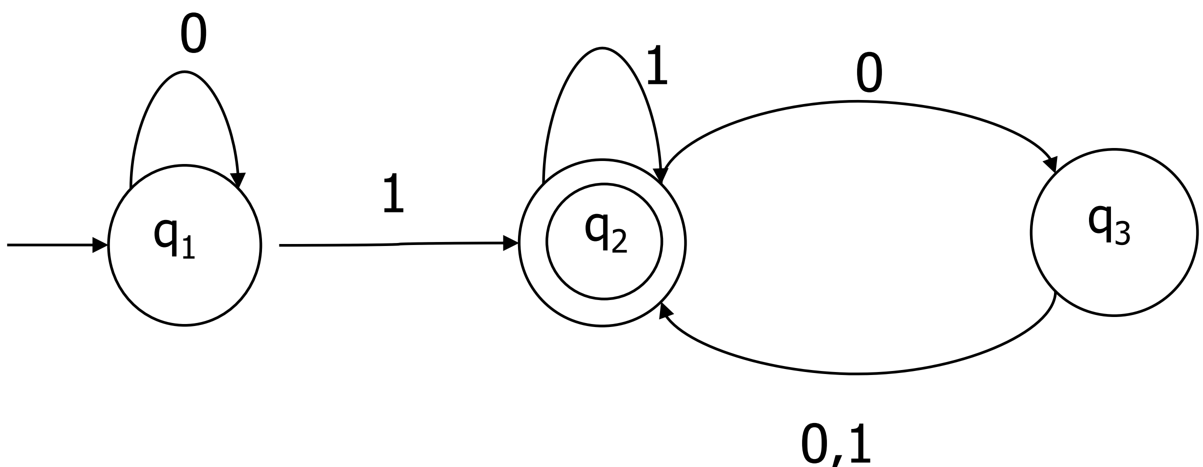
- So far, every step of a computation follows in a unique way from the preceding step.
- When the machine is in a given state and reads the next input symbol, we know what the next state will be – it is called **deterministic computation**
- **Deterministic Finite Automata -- DFA**

Nondeterminism

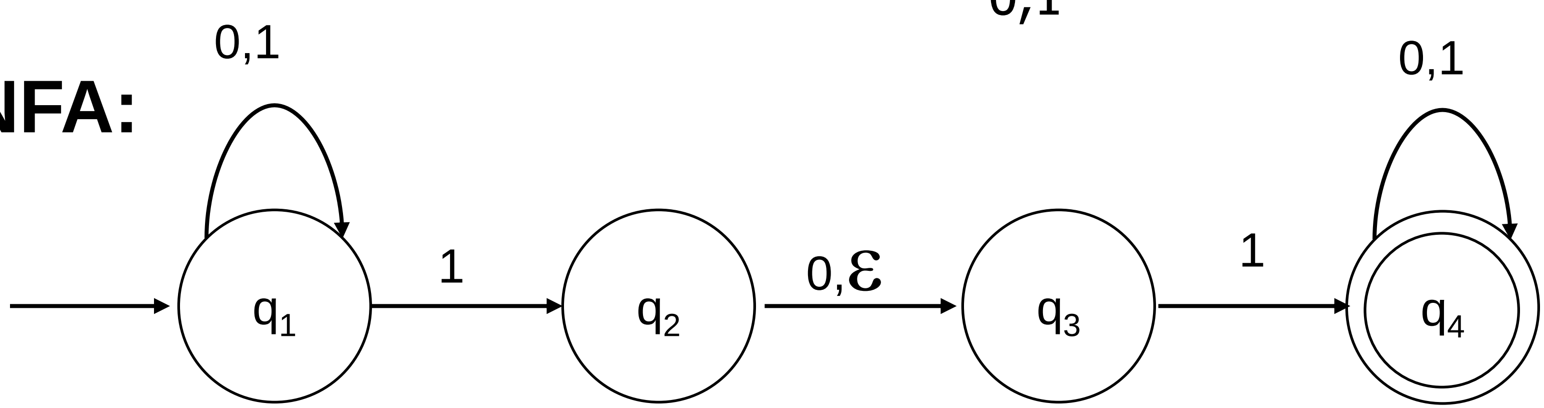
- In a nondeterministic machine, several choices may exist for the next state at any point.
- Nondeterminism is a generalization of the determinism, so every deterministic finite automaton is automatically a nondeterministic finite automaton.
- **Nondeterministic Finite Automata--NFA**

Example of DFA vs. NFA

DFA:



NFA:



Differences between DFA & NFA

- Every state of DFA always has exactly one exiting transition arrow for each symbol in the alphabet while the NFA can violate the rule.
- In a DFA, labels on the transition arrows are from the alphabet while NFA can have an arrow with the label ε.

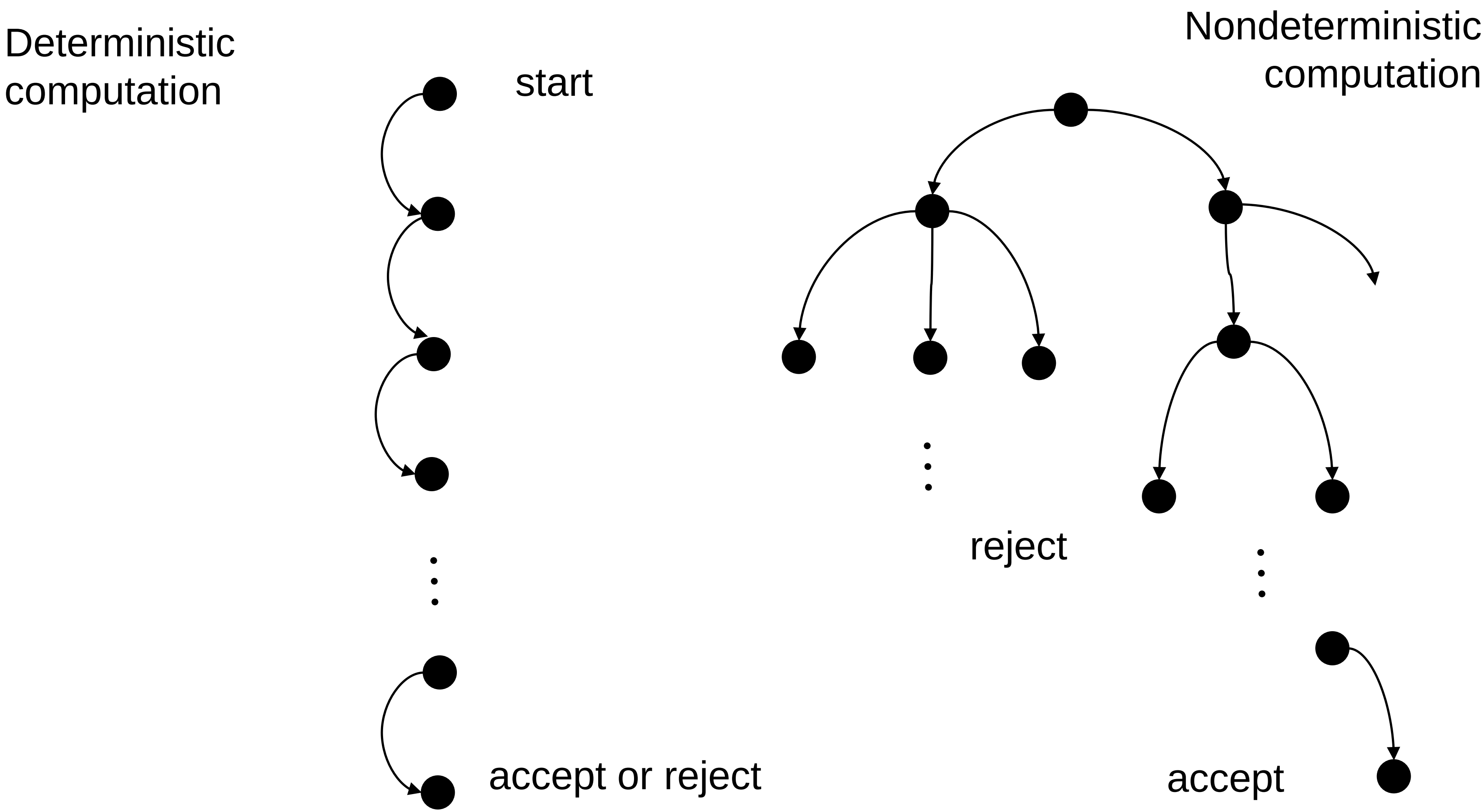
How does the NFA work?

- When we are at a state with multiple choices to proceed (including ε symbol), the machine splits into multiple copies of itself and follow all the possibilities in parallel.
- Each copy of the machine takes one of possible ways to proceed and continuous as before. If there are subsequent choices, the machine splits again.
- If the next input symbol doesn't appear on any of the arrows exiting the state occupied by a copy of the machine, that copy dies.
- If any one of these copies is in an accept state at the end of the input, the NFA accepts the input string.

Tree of possibilities

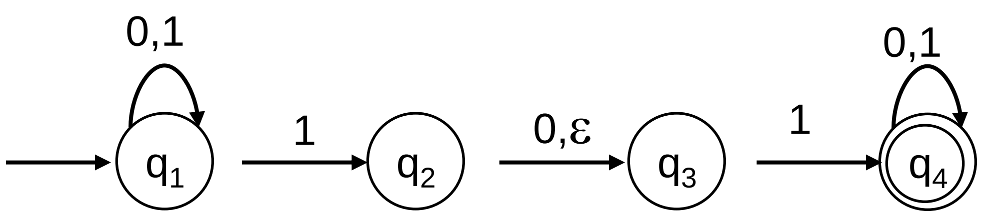
- Think of a nondeterministic computation as a tree of possibilities
- The root of the tree corresponds to the start of the computation.
- Every branch point in the tree corresponds to a point in the computation at which the machine has multiple choices.
- The machine accepts if at least one of the computation branches ends in the an accept state.

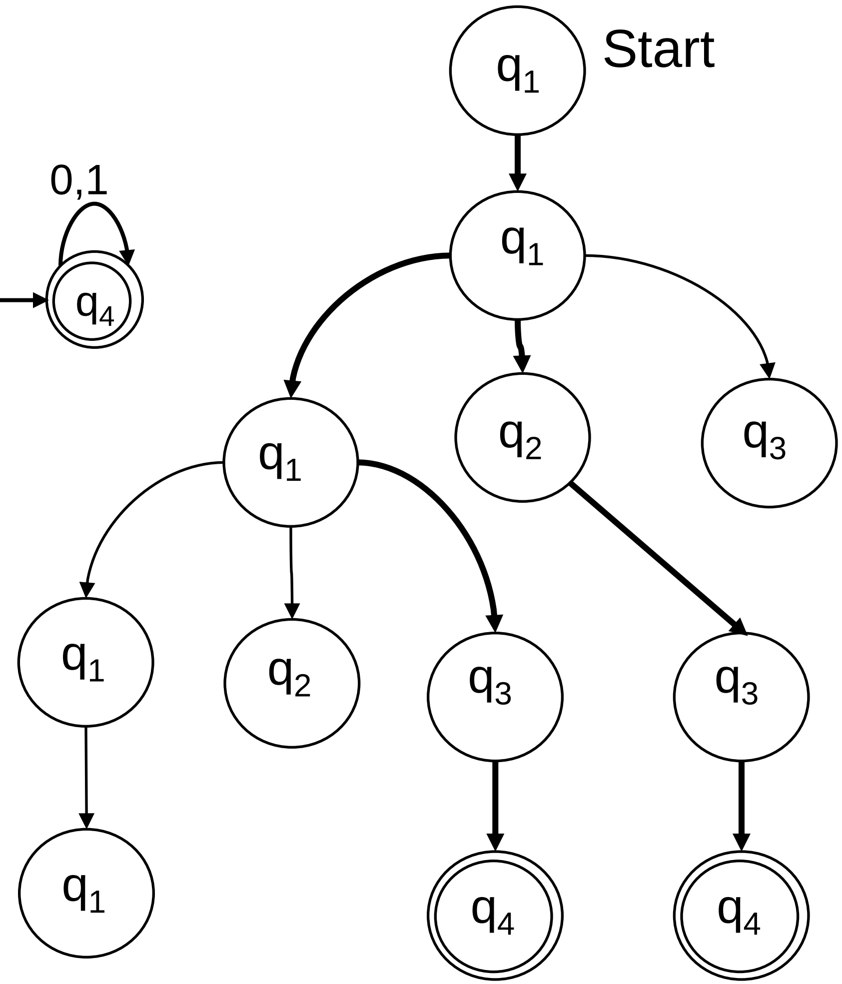
Tree of possibilities



Example: 010110

NFA:





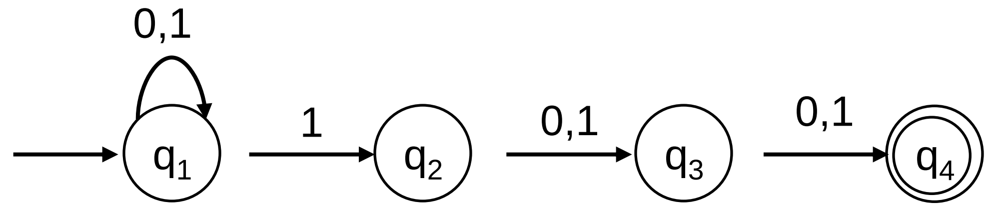
Properites of NFA

- Every NFA can be converted into an equivalent DFA.
- Constructing NFAs is sometimes easier than directly construction DFAs.
- NFA may be much smaller than it DFA counterpart.
- NFA’s functioning may be easier to understand.
- Good introduction to nondeterminism in more powerful computational models because FA are especially easy to understand.

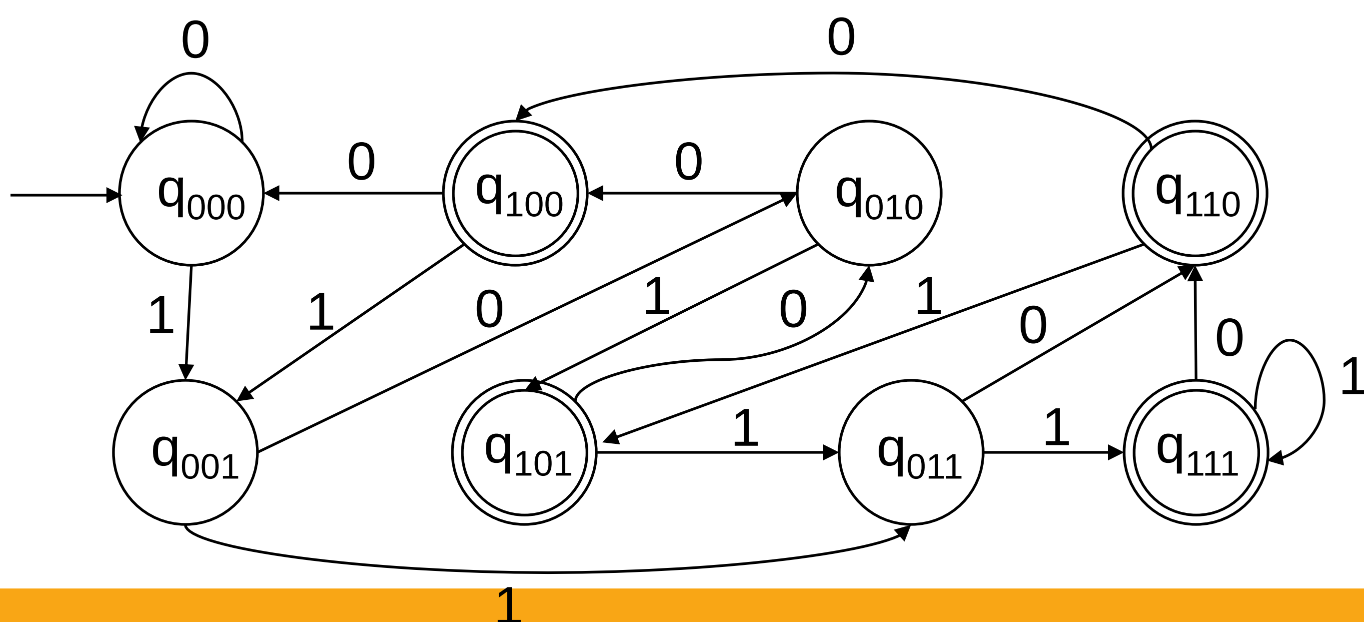
ocw.utm.my

Example:
Converting NFA into DFA

NFA: recognizes language which contains 1 in the third position from the end



Equivalent DFA:



Formal definition of NFA

- A nondeterministic finite automaton is a 5-tuple (Q,Σ,δ,q_0,F) , where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta : Q \times \Sigma_\varepsilon \rightarrow P(Q)$ is the transition function,
  - q_0 is the start state, and
  - $F \subseteq Q$ is the set of accept states.

Notation:

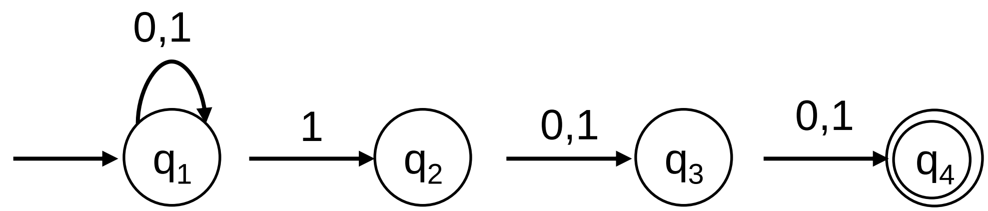
P(Q) is called power set of Q (a collection of all subsets of Q).

and $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

ocw.utm.my

Example: Formal definition of NFA

NFA N_1:



- Formal definition of N_1 is (Q,Σ,δ,q_0,F) , where
  - $Q = \{q_1,q_2,q_3,q_4\}$
  - $\Sigma = \{0,1\}$
  - δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1,q_2\}$	$\varnothing$
q_2	$\{q_3\}$	$\varnothing$	$\{q_3\}$
q_3	$\varnothing$	$\{q_4\}$	$\varnothing$
q_4	$\{q_4\}$	$\{q_4\}$	$\varnothing$

  - q_0 is the start state, and
  - $F = \{q_4\}$

regular expressions

- The new definition we have talked about is claimed as "Regular Expression".
- Languages which are able to be described by RE, are called "Regular Languages".
  - Not every languages are able to be described by RE.
  - Regular languages may also be described by another fine definitions, besides the RE.

- The symbols that appear in RE are
  - the letters of the alphabet Σ
  - the symbol of null string ε or λ
  - parentheses ( )
  - star operator *
  - $\cup$ or + sign

regular expressions

Say that R is a regular expression if R is

1. a for some a in the alphabet Σ,
2. ε or λ,
3. $\varnothing$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is regular expression.

- Rule 1: Every letter of Σ can be made into a regular expression
- Rule 2: If $\mathbf{r_1}$ and $\mathbf{r_2}$ are regular expressions, then so are
  - (i) $(\mathbf{r_1})$
  - (ii) $\mathbf{r_1 r_2}$ **or** $\mathbf{r_1 \circ r_2}$
  - (iii) $\mathbf{r_1 \cup r_2}$
  - (iv) $\mathbf{r_1^*}$
- Rule 3: Nothing else is a regular expression.

regular set

- **Basis** : ϕ, $\{ \lambda \}$ and $\{a\}$, for every $a \in \Sigma$ are regular sets over Σ.
- **Recursive step**: Assume X and Y are regular set over Σ , then the sets:

  $X \cup Y$, XY and X^*

  are regular sets over Σ
- **Closure**: X is a regular set over Σ iff it can be obtained from basis elements by a finite number of applications of ecursive step.

- We use parentheses ( ) as an option to eliminate the ambiguity when we apply * or $^+$ to the expressions. For example:

if $\mathbf{r_1 = aa \cup b}$ then what is $\mathbf{r_1^*}$?

Is the $\mathbf{r_1^* = aa+b^*}$ or $\mathbf{(aa+b)^*}$?

They are both REs but very different.

Ans. the later choice. In this case we should put the ( ) when we substitute $\mathbf{aa \cup b}$ to $\mathbf{r_1^*}$

- ε or λ is the symbol of null string in regular expression.
- $\varnothing$ is the symbol for "Null Language"
- Don't confuse!
  - R = λ represents the language containing a single string, the empty string. ⇨ $\{\lambda\}$
  - R = $\varnothing$ represents the language that doesn't contain any strings.

- If we let R be any regular expression,
  - $R \cup \varnothing = R$:

    Adding the empty language to any other language will not change it.
  - $R \circ \varepsilon = R$:

    Adding the empty string to any other language will not change it.
  - $R \cup \varepsilon$ may not equal to R

    e.g., if $R = 0$, the $L(R) = \{0\}$ but $L(R \cup \varepsilon) = \{0, \varepsilon\}$
  - $R \circ \varnothing$ may not equal to R

    e.g., if $R = 0$, the $L(R) = \{0\}$ but $L(R \circ \varnothing) = \varnothing$

- Let consider the language defined by

$$(a \cup b)^* a (a \cup b)^*$$

What does it produce ?

Ans.

The language which is the set of all words over the alphabet $\Sigma = \{a,b\}$ that have an a in somewhere.

Only words which are not in this language are those that have only b's and the word ε

- Those words which compose of only b's are defined by the expression $\mathbf{b}^*$.

(b* also includes the null string ε)

- Therefore, the language of all strings over the alphabet $\Sigma = \{a,b\}$ are

all strings = (all strings with an a) ∪ (all string without an a)

$$(\mathbf{a}\cup\mathbf{b})^* = (\mathbf{a}\cup\mathbf{b})^*\mathbf{a}\,(\mathbf{a}\cup\mathbf{b})^* \cup \mathbf{b}^*$$

- How can we describe the language of all words that have at least 2 a's ?

Ans

$$(\mathbf{a} \cup \mathbf{b})^* \mathbf{a} (\mathbf{a} \cup \mathbf{b})^* \mathbf{a} (\mathbf{a} \cup \mathbf{b})^*$$

= (some beginning)(the first a)(some middle)(the second a)(some end)

where the arbitrary parts can have as many a's (or b's) as they want.

- Is there any other RE that can define the language with at least 2 a's ?

Ans. Yes. For example:

$$\mathbf{b}^*\mathbf{a}\mathbf{b}^*\mathbf{a}(\mathbf{a}\cup\mathbf{b})^*$$

=(some beginning of b's (if any))(the first a) (some middle of b's)(the second a) (some end)

$(\mathbf{a} \cup \mathbf{b})^* \mathbf{a} (\mathbf{a} \cup \mathbf{b})^* \mathbf{a} (\mathbf{a} \cup \mathbf{b})^* = \mathbf{b}^* \mathbf{a} \mathbf{b}^* \mathbf{a} (\mathbf{a} \cup \mathbf{b})^*$

Both expressions are **equivalent** because they both describe the same item. We could write

language $((\mathbf{a} \cup \mathbf{b})^* \mathbf{a} (\mathbf{a} \cup \mathbf{b})^* \mathbf{a} (\mathbf{a} \cup \mathbf{b})^*)$

= language$(\mathbf{b}^* \mathbf{a} \mathbf{b}^* \mathbf{a} (\mathbf{a} \cup \mathbf{b})^*)$

= all words with at least two a's

$= (\mathbf{a} \cup \mathbf{b})^* \mathbf{a} \mathbf{b}^* \mathbf{a} \mathbf{b}^*$

$= \mathbf{b}^* \mathbf{a} (\mathbf{a} \cup \mathbf{b})^* \mathbf{a} \mathbf{b}^*$

- If we wanted all words with exactly 2 a's, we could use the expression

$$b^*ab^*ab^*$$

it can describes such words as

aab, baba, bbbabbbab, ...

Question: Can it make the word aab ?

Ans : Yes, by having the first and second

$$b^* = \lambda$$

- How about the language with at least one a and at least one b ?

$$(a\cup b)^*a\,(a\cup b)^*b\,(a\cup b)^*$$

It can only produce words which an a precede ab. To produce words which have ab precede an a, we can describe by

$$(a\cup b)^*b(a\cup b)^*a\,(a\cup b)^*$$

Thus, the set of all words :

$$(a\cup b)^*a\,(a\cup b)^*b\,(a\cup b)^* \cup (a\cup b)^*b\,(a\cup b)^*a\,(a\cup b)^*$$

- $(a \cup b)^*a\,(a \cup b)^*b\,(a \cup b)^*$ can produce all words with at least one a and at least one b,
- However, it doesn't contain the words of the forms some b's followed by some a's.
- These exceptions are all defined by bb^*aa^*
- Thus, we have all strings over $\Sigma = \{a,b\}$

$$\mathbf{(a \cup b)^*a\,(a \cup b)^*b\,(a \cup b)^* \cup (a \cup b)^*b\,(a \cup b)^*a\,(a \cup b)^*}$$

$$= (a \cup b)^*a\,(a \cup b)^*b\,(a \cup b)^* \cup bb^*aa^*$$

$$(a\cup b)^*a\,(a\cup b)^*b\,(a\cup b)^* \cup bb^*aa^*$$

- generates all words which have both a and b in them somewhere.
- Words which are not included in the above expression are words of all a's, all b's or ε ⇨ a^*, b^*
- Now, we have all words which can be generated above the alphabet

$$(a\cup b)^* = (a\cup b)^*a\,(a\cup b)^*b\,(a\cup b)^* \cup bb^*aa^* \cup a^* \cup b^*$$

References