Chapter 7. Automata

1. 형식 언어와 오토마타
   1.1 언어
   1.2 언어와 문법

2. Automata
   2.1 Definitions
   2.2 Membership in a special class
   2.3 FSA as a Language Recognizer

3. Turing Machine
   3.1 Definitions
   3.2 Examples

1.1 언어 (Language)

- S : 기호들의 집합
- S^*: S로부터 만들어지는 모든 유한 스트링들

예) S: 알파벳
S^*: 가능한 모든 영어 문장들

Ex) S= {정수, +, -, x, ÷, (,)}
S^*: 모든 가능한 수식들(algebraic expressions)

언어: 다음의 세가지 요소로 구성된다.
(1)기호들(알파벳)의 집합 **S**가 반드시 존재한다.

(2)S로부터 문장들의 집합 $\mathbf{S}^*$ 를 형성하는 규칙이 반드시 존재한다.
(syntax - specification of the proper construction of sentence**)**

(3) 규칙에 합당하게 만들어진 문장들이 어떤 의미를 갖는지를 반드시 결정할 수 있어야 한다.
(semantics - the specification of the meaning of sentences**)**

1.2 언어와 문법

- 인류의 언어 – 자연언어
- 프로그래밍언어 - 컴퓨터의 발전과 함께 다양화, 고급화.
  (프로그램언어설계의 이론적 배경 – 형식 언어 이론)
- 형식 언어 이론 (Formal language theory) – 프로그램언어는 자연언어의 범주로 해석 가능

형식언어 이론에서 문법: **G=(N, T, S, P)**

N: 비 단말(nonterminal)기호의 집합. T: 단말기호(terminal symbol)

S: 시작기호(starting symbol),

P: 생성규칙(production rule, $\Rightarrow$) If w $\Rightarrow$w', w is replaced by w'

- 문법 정의 → 생성규칙의 반복적 적용 → 열(string) 생성 (단말, 비 단말 혼합 됨) → 최종 단말기호들 (문장 – sentence)

예: $N=\{S\}$, $T=\{a, b, A\}$, $V= T \cup N$, $P = \{S \Rightarrow aA, S \Rightarrow b, A \Rightarrow aa\}$

생성규칙 $S \Rightarrow aA$ 와 $A \Rightarrow aa$를 이용, **aaa 유도**, $S \Rightarrow$ b로부터 **b를 유도**.

따라서 문법 G로부터 유도되는 언어는 $L(G)=\{b, aaa\}$ 이다.

구문법(phase-structure grammar)

예) T={John, Jill, drives, jogs, carelessly, rapidly, frequently}

N={sentence, noun, verbphrase, verb, adverb}, V = T∪N,

P: <sentence> ⇒ <noun> <verbphrase>, <noun> ⇒ John <noun> ⇒ Jill

<verbphrase> ⇒ <verb><adverb>, <verb> ⇒ drives <verb> ⇒ jogs

<adverb> ⇒carelessly <adverb> ⇒rapidly <adverb> ⇒frequently

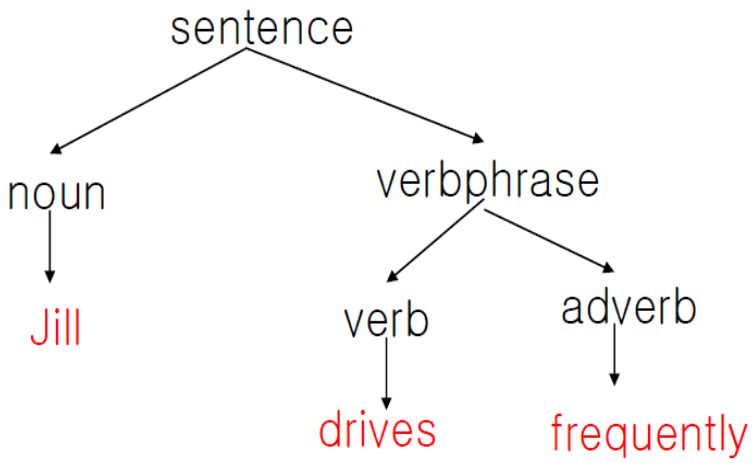
Q: "**Jill drives frequently**."는 문법적으로 맞는 문장인가?

<sentence> ⇒ <noun> <verbphrase>

⇒ Jill <verb><adverb>

⇒ Jill drives frequently

• derivation tree (parse tree)



문법의 표현

- BNF(Backus-Naur Form)형식 .

(1) 비 단말 기호는 <a>로 표시한다.
(2) 생성 ⇒은 ::=로 표시한다.
(3) 하나의 비 단말 기호로 부터 생성되는 여러 문자열은 '|' 으로 구분한다.

예: C언어에서 사용되는 signed integer를 표기하기 위한 생성 규칙을 BNF로 표시하면.

<signed integer> ::= <sign><integer>
<sign> ::= +|−
<integer> ::= <digit>|<digit><integer>
<digit> ::= 0|1|2|3|4|5|6|7|8|9

ex) 프로그래밍 언어의 산술문, **id * (id + number)**

N ={expression, term, factor},

T ={+, -, *, /, (,), id, number}

P: <expression> → <expression> + <term> |
<expression> - <term> | <term>

<term> → <term> * <factor> | <term> / <factor> | <factor>

<factor> → id | number | (<expression>)

생성과정 :

<expression> → <term> → <term> * <factor>

→ <factor> * <factor> → id * <factor> → id * (<expression>)

→ id * (<expression> + <term>)

→ id * (<term> + <term>)

→ id * (<factor> + <term>)

→ id * (id + <term>)

→ id * (id + <factor>)

→ **id * (id + number)**

Chomsky's 4 Grammar

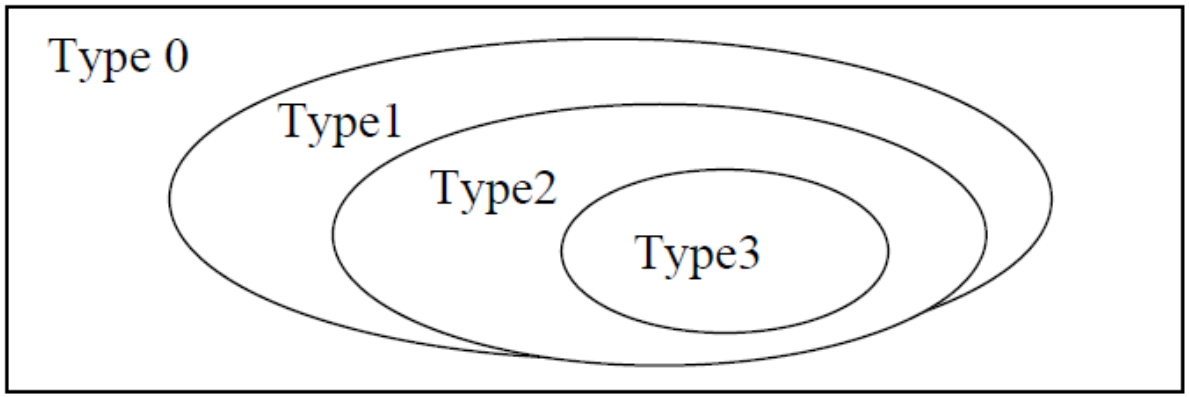
- 문법의 제약: 생성규칙의 제약된 형태에 따라 네 등급(4 class)으로 분류.

TYPE0: Unrestricted grammar (recursively enumerable set) **Turing Machine**

TYPE1: Context-sensitive G. (Context-sensitive Lang.) Linear bounded **Automata**

TYPE2: Context-Free G. (Context-Free Lang.) Push-Down **Automata**

TYPE3: Regular (Regular Lang) **FSA (finite state automata)**



2. Automata – Theory of Computation

Finite state machine(FSM)/Finite State Automata(FSA)

- Abstract model of a machine with a memory
- Mathematical model with finite set of **inputs** and goes through set of **states** and may have **outputs** (Computational models)
- For theoretical modeling capability=>**Turing Machine(TM)** is most capable class of automata

. 오토마타이론은 계산능력이 있는 추상기계와 그 기계를 이용해서 풀 수 있는 문제들을 연구하는 컴퓨터공학의 분야.

. 유한한 **상태**를 가지고, **입력**에 따라 **상태를 전이하며**, **출력** 한다.
➔ 알고리즘이 요구하는 계산문제를 해결하는 능력과 같다.

* 계산문제는 추상기계와 형식언어, 형식문법은 불가분의 관계가 된다.
따라서 오토마타는 언어와 문법과 같은 계층 분류를 가진다

- 오토마타는 이산 시간 동안 주어진 입력에 의존해 작동하는 수학적인 기계이다.

 - 기계는 일정 주기마다 입력을 하나씩 받는데, 이를 기호 또는 문자라고 한다.

- 기계가 입력 받는 문자는 정해진 집합의 한 원소 이어야 하며, 이를 알파벳이라 한다.

- 기계가 입력 받는 일련의 기호와 문자를 문자열이라 한다.

- 기계는 유한한 **상태(state)**의 집합을 갖고 있으며, 입력에 따라 **현재 상태에서 다음 상태로** 전이한다.

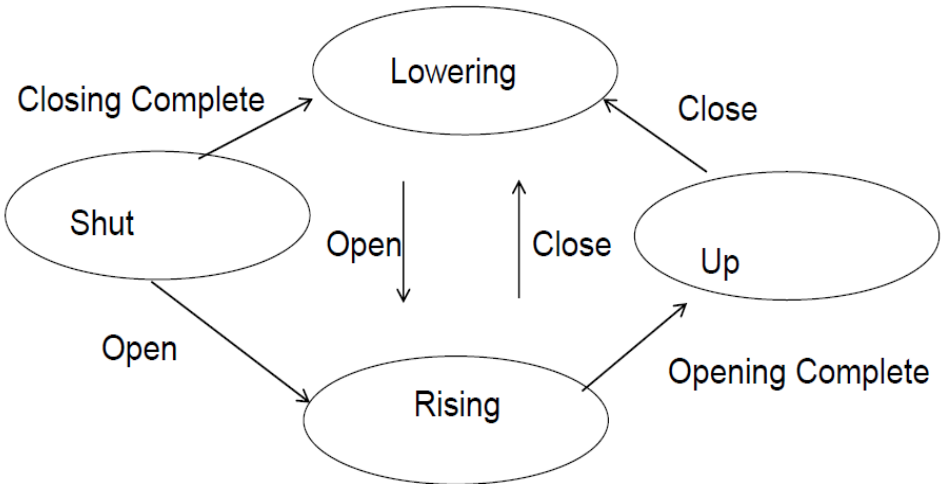
- 기계는 입력의 끝을 만나거나 특정 상태에 있을 때 **정지(halt)할** 수 있다. 기계는 정지했을 때 문자열을 **수용/거부한다**.

Application

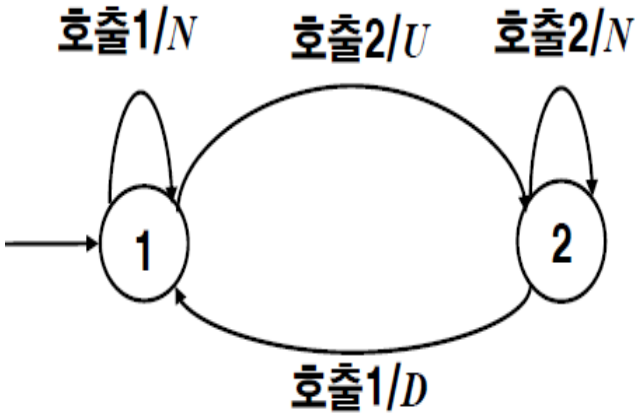
- 오토마타는 컴퓨터 구조설계와 컴파일러 설계(design), 파싱(parsing), 정형 모델의 검증(verification of formal model) 등의 중요한 요소이다.

- Application Example: Vending machine, coin changer, elevators, compilers, network protocol, switching circuit

ex) Garage Door (차고 문) State Diagram



ex) 1층과 2층만 운행하는 승강기



Ex) 자판기 문제: 100, 500원 동전으로, 700원 입력 되면, C, S버튼 입력 시 콜라, 사이다 나옴. (초과금액은 반납)

입력: (100, 500, C, S) 상태: (s0, s1, s2, s3, s4, s5, s6, s7)

출력: (n, 100, 200, 300, 400, 500, 콜라(c), 사이다(s))

	다음 상태				출력			
상태	100	500	C	S	100	500	C	S
S_0	S_1	S_5	S_0	S_0	n	n	n	n
S_1	S_2	S_6	S_1	S_1	n	n	n	n
S_2	S_3	S_7	S_2	S_2	n	n	n	n
S_3	S_4	S_7	S_3	S_3	n	100	n	n
S_4	S_5	S_7	S_4	S_4	n	200	n	n
S_5	S_6	S_7	S_5	S_5	n	300	n	n
S_6	S_7	S_7	S_6	S_6	n	400	n	n
S_7	S_7	S_7	S_0	S_0	100	500	C	S

* Definition of FSA

M = (S, A, T, S_0, F)

S: set of states, A: input symbols

T: transitions, S_0: start/initial state

F: final/end/terminate/accept state

ex) **M** may be the automata with Alphabet

S = {0,1,2,3}

A = {a,b}

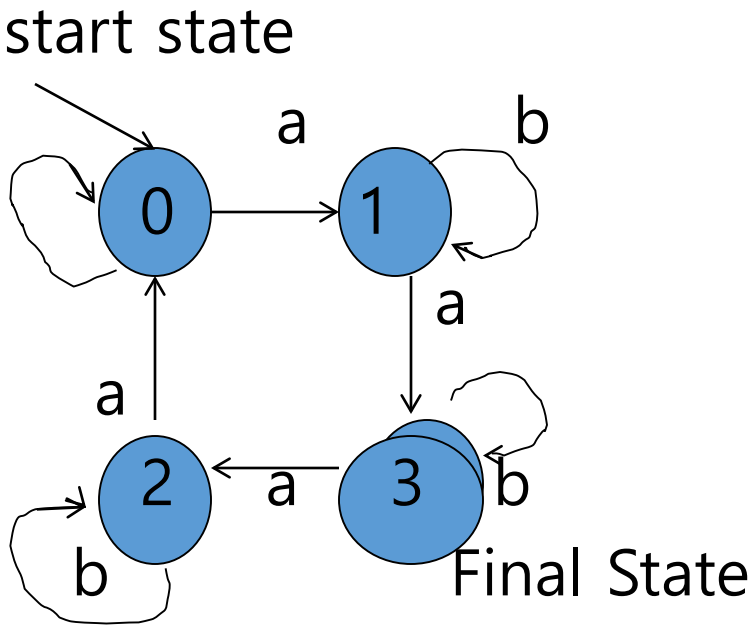
S_0 = 0

F = {3}

		a	b
states	0	1	0
	1	3	1
	2	0	2
	3	2	3

Transition function T

Question: Give a representation of the automata as a labeled digraph (**State Diagram**), and describe the action of the automata



	a	b
0	0	1
1	0	2
2	0	2

- FSA – Without output, With output

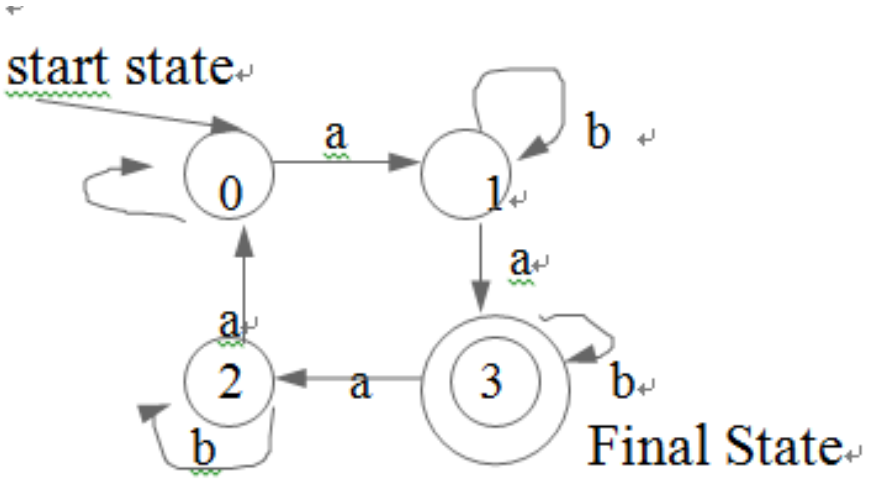
FSA without output (membership in a special class)

- Rule: Begin from start state.

  Transition => state to state according to input

  Ends with final state

Ex) Recognizer



Abaa => 0,1,1,3,2 ; does not HALT. Not valid input

Aba => 0,1,3; HALT (accept input 'aba')

abaaaaa => 0,1,1,3,2,0,1,3 ; HALT (accept input)

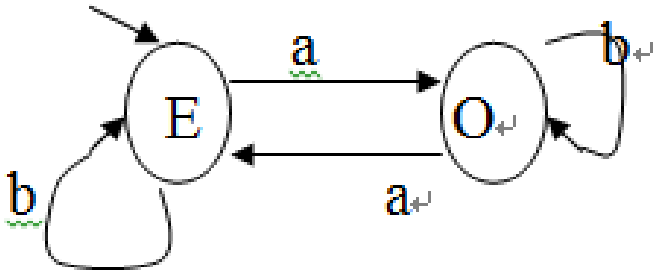
➔ This automata will HALT iff input sequence contains K개의 a's when k is 4n+2, $n \geq 0$.

➔ n=0, aa; n=1, aaaaaa; (don't care b), n=3, …..;

Ex) Design a FSA that accepts strings over {a,b} that contains odd number of a's

- Algorithm:

{ Input: n (length of string, S_1, S_2 ,....S_n); Output: Accept, Reject}




```
Procedure FSA(s,n)
  State = 'E'
  For I=1 to n do
  Begin
    If state = 'E' and Si='a' then state ='O'
    If state ='O' and Si='a' then state = 'E'
 End
 If state = 'O' then return('Accept')
 Else   return('Reject')
End FSA
```

Design Automata

- Design a FSA that accepts strings over {a,b} that accepts even number of a's
- Design a FSA that accepts strings over {a,b}, such that "the number of b's is divisible by 3"
- Design a FSA that accepts strings over {a,b}, such that "the number of b's is divisible by 3 and the number of a's is even"
- 'aabb'를 substring으로 가진 string만 인식하는 FSA를 design하시오 (A = {a,b})

* FSA as a Language recognizer

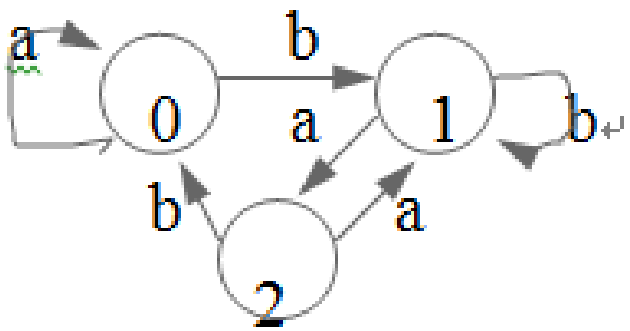
- Question : If L is a language over A and x is a word in A*, then is x a member of L?

(pf) Let (S, A, T, S_0, F) be an FSA

Define extension of T to T′, recursively

1) $T'(S, \lambda) = S, \quad \forall s \in S$
2) $T'(S, ax) = T'(T(S, a), x), \quad \forall s \in S', x \in A^*, a \in A$

ex) Compute T'(0, abba) for FSA given transition



	a	b
0	0	1
1	2	1
2	1	0

T'(0, abba) =T'(T(0,a), bba)

= T'(0,bba) = T'(T(0,b), ba) = T'(1,ba)

= T'(T(1,b), a)= T'(1,a) = 2

S={0,1,2} A={a,b}

0-> initial state 2-> final state

Halting

1) FSA **halts** when all symbols are acted on
2) If FSA **halts** in one of final state => the word is accepted/recognized by FSA

- Language accepted by FSA is set of words L(M), given by $L(M) = \{x | x \in A^*, \text{ and } T'(S_0, x) \in F\}$

ex) Find language L(M) accepted by FSA M given by table

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

$S_0 = 0$

$F = \{2\}$

(sol) $L(M) = b^n(ab)^m$, $n \geq 0$, $m \geq 1$

ex) ab, bab, bbbab,... bbababab,,,

exercise

ex) Find FSA that accepts all words in {0,1}*,

end in 0 and have length ≤3

ex) FSA for unsigned fixed-point numbers (real number) without trailing or leading zero (any problem??)

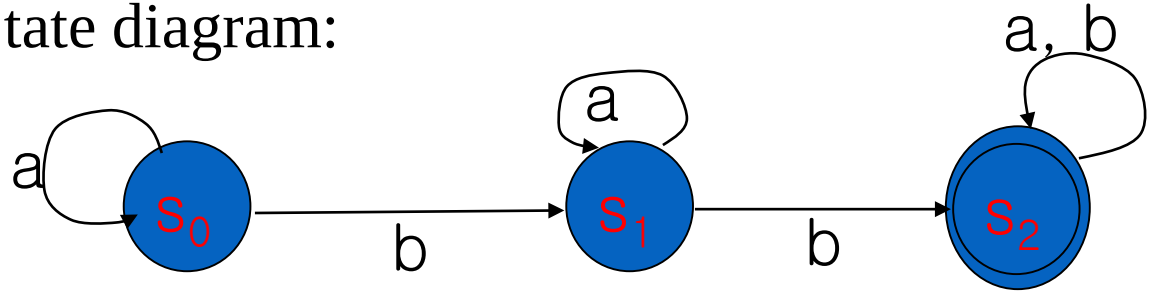
ex) Design an FSA to recognize identifier =>

space + letter/digits + space (first=letter)

Ex) More examples

Input symbol = {a, b} state = {s0, s1, s2}

State diagram:



Transition:

	a	b
s0		
s1		
s2		

L(M) => L(G)

G: regular grammar

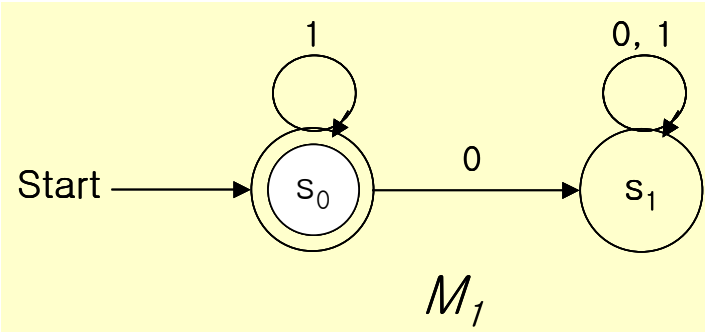
T={a,b} N={s0, s1, s2}

Production rule:

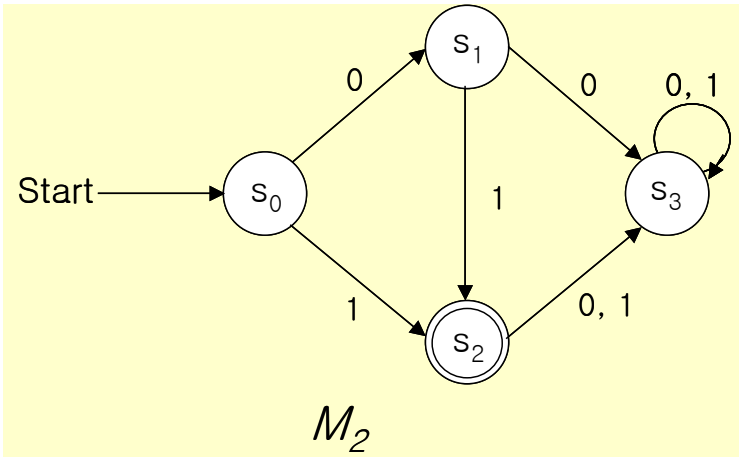
$S_0 \Rightarrow aS_0$
$S_0 \Rightarrow bS_1$
$S_1 \Rightarrow aS_1$
$S_1 \Rightarrow bS_2$
$S_2 \Rightarrow aS_2$
$S_2 \Rightarrow bS_2$
$S_1 \Rightarrow b$
$S_2 \Rightarrow a$
$S_2 \Rightarrow b$

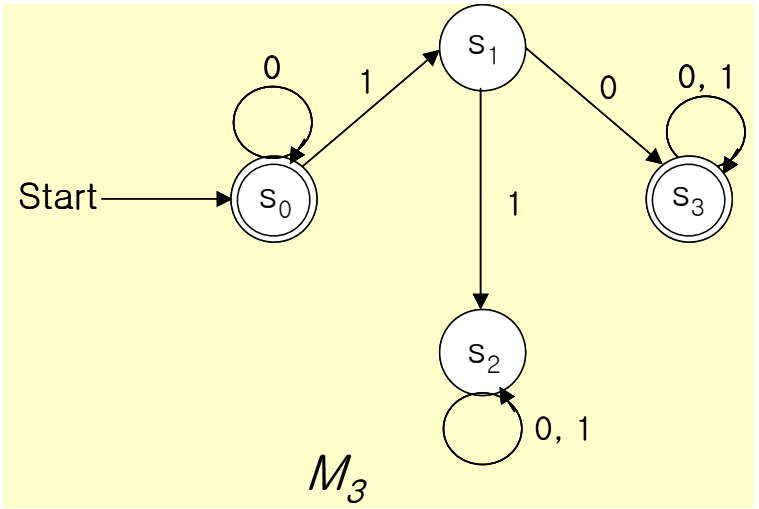
1) 다음의 FSM M1에 의해 인식되는 언어 L(M1)을 구하라

$L(M1) = \{1^n \mid n = 0,1, \ldots\}$



2) FSM M_2에 의해 인식되는 언어 $L(M_2)$을 구하라 => $L(M_2) = \{1, 01\}$





3) FSM M_3에 의해 인식되는 언어 $L(M_3)$을 구하라

$L(M_3) = \{0^n, \ 0^n10X \mid n = 0, 1, \ldots,$ and X는 0과 1로 된 임의의 문자열$\}$

FSA with Output

- FSA without Output => follow the path and accept or do not accept used as recognizer

- FSA with output = FSM (concerns about generated output)

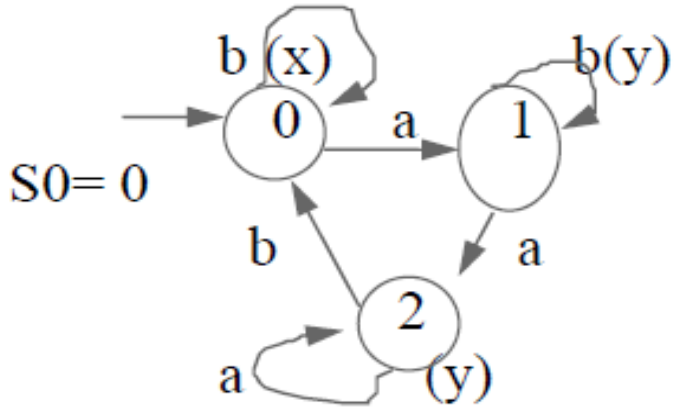
- Moore Machine

  M = (S, A, T, O, f) O: output alphabet,

  f: S->O output function

ex) Input = abbaaa determine output

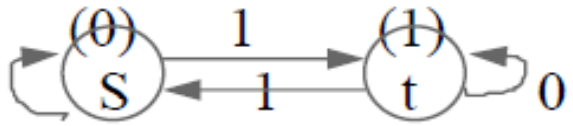
	I (a	b)	O
0	1	0	x
1	2	1	y
2	2	0	y



I	a	b	b	a	a	a	-
S	O	1	1	1	2	2	2
O	x	y	y	y	y	y	y

output: xyyyyyy

ex)



S	I(0	1)	O
s	s	t	0
t	t	s	1

question: Find output for 001 => 0001

10101 => 011001

- 문제: 다음의 언어는 정규 언어인가?

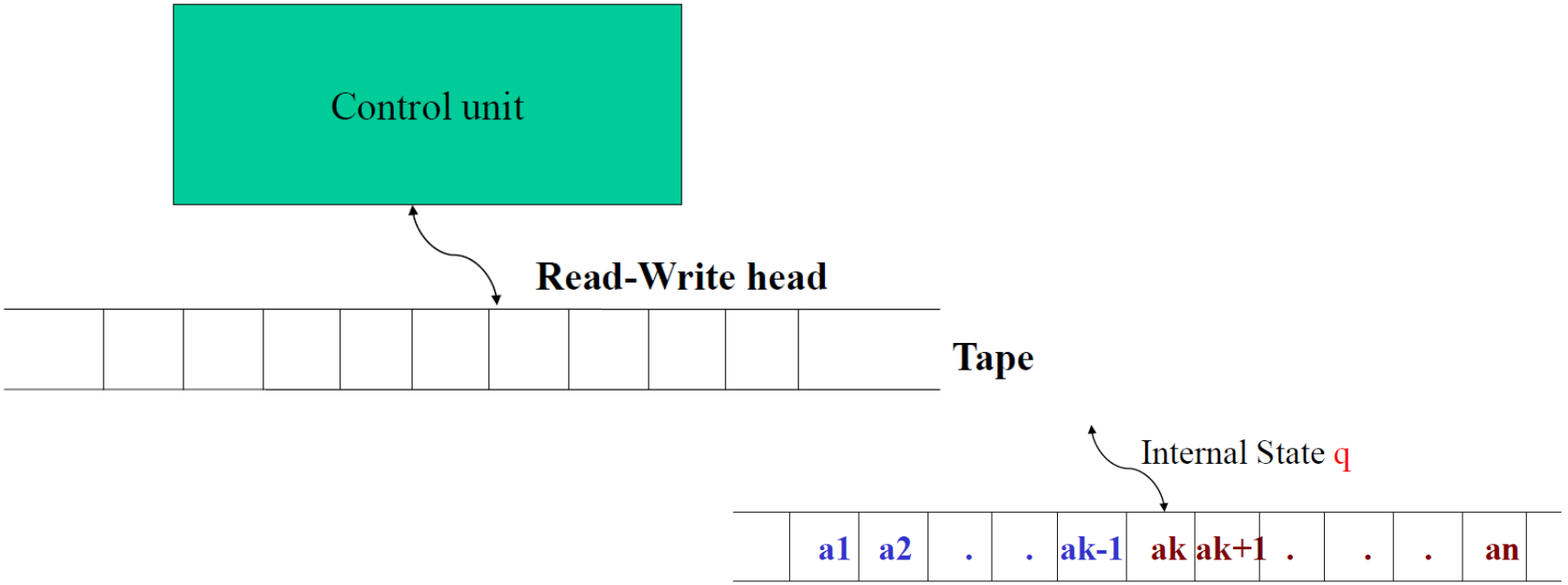
$$L = \{x^n y^n \mid n = 1,2,3,\ldots\}$$

  - 정규식으로 L을 표현할 수 있을까? No.
  - 따라서 위의 언어를 인식하는 유한상태 오토마타를 만들 수 없다.
  - 위의 언어를 인식하기 위해서 튜링 기계(Turing machine)라고 하는 보다 강력한 기계를 사용할 수 있다.

- 최적화: 이와 같이 특정 과제를 수행하는 유한상태 오토마타를 설계할 수 있지만 최적의 기계는 아니다. 즉, 더 적은 상태의 수를 갖는 유한상태 오토마타를 설계할 수 있으며, 다음 단계에서 필요한 것은 최적화된 유한상태 오토마타의 설계이다.

Turing Machine

- Turing's Computation (**Alan Tutoring**)
  . Model of computation – **Turing Machine** (1936)
  . Encode each input with finite symbols place them on "infinite tape"
  to be read/acted by human computer
  . scan one symbol at a time, move - one call to the Left or Right



- a1a2…ak-1qakak+1…an

Definition of TM

- Define TM as 5-tuple (S, A_t, N, S_0, F)

  S: finite set (states) $S_0 \in S$, $F \subseteq S$

  A_t: finite tape alphabet

  (A_i: input, Ao: output Δ: blank)

  N: $S \times A_t \rightarrow S \times A_t \times \{L, R\}$ next mode function

  (L: Left R: right)

- λ: null word (left end of tape, last nonblank symbols)

- Difference with FSA

  . FSA may not rewrite, . Can't look back symbols

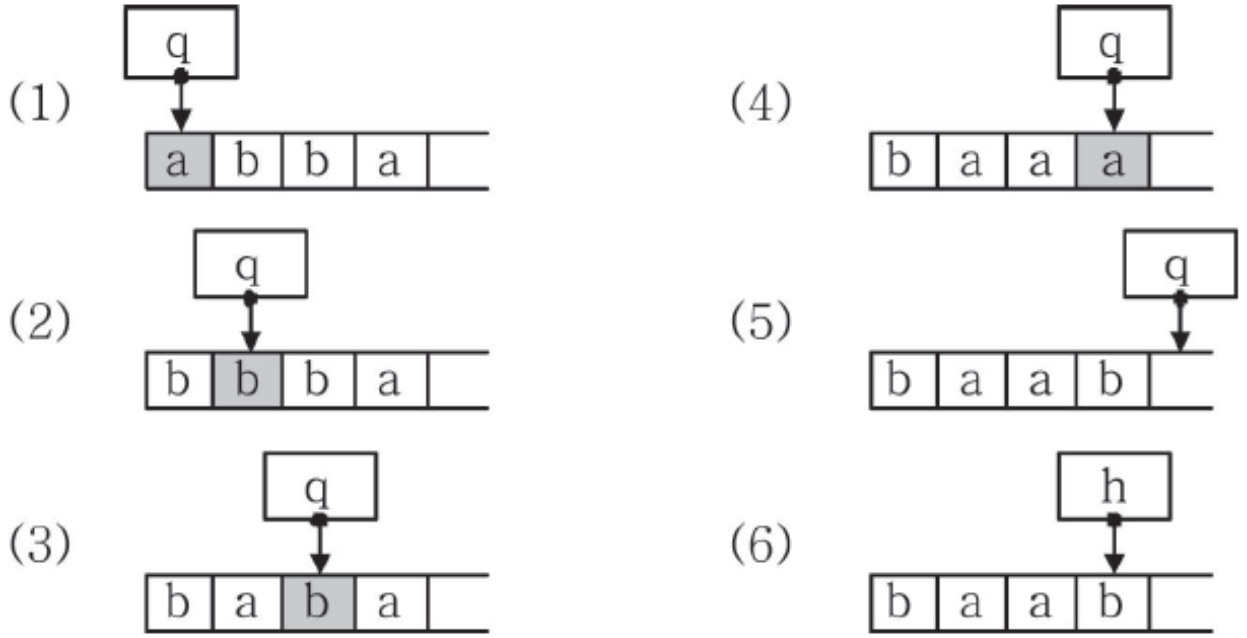
  . partial transition(TM)<-> complete transition (FSA)

Example

- a와b의 교환

- State transition

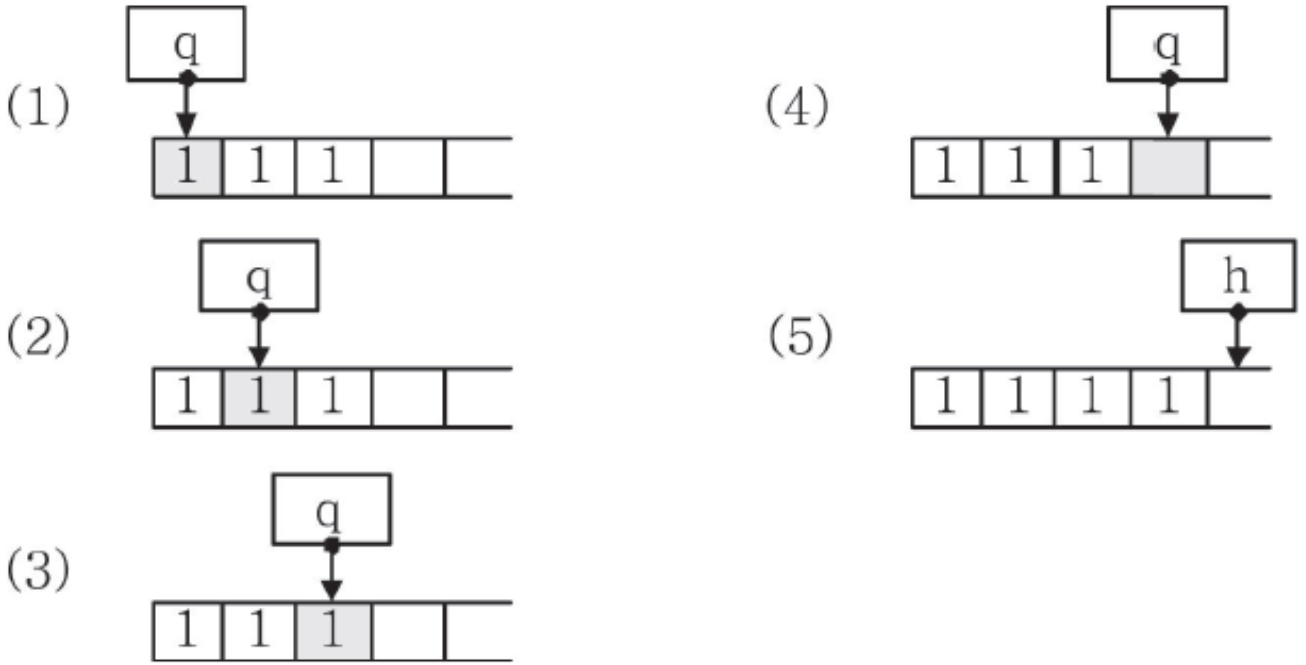
	a	b	λ
q	(q,b,R)	(q,a,R)	(h, λ, L)
h	-	-	-



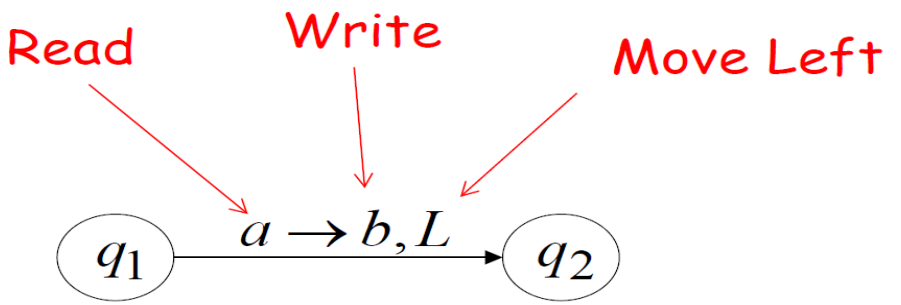
Example

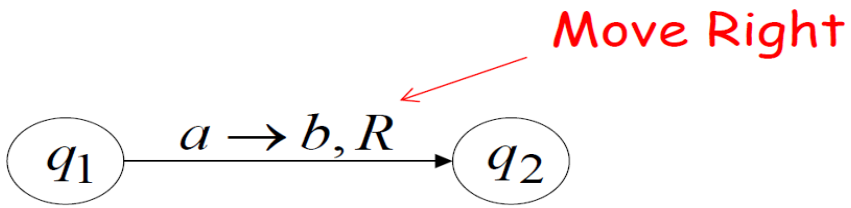
f(n)= n+1 계산, M = ({q,h}, {1, λ}, {h}, q) Input: 111

	1	λ
q	(q, 1, R)	(h, 1, R)
h	-	-



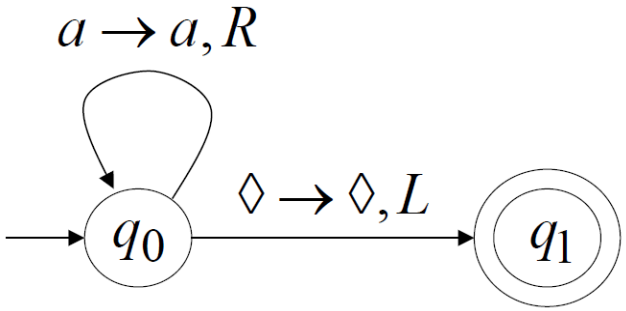
States & Transitions





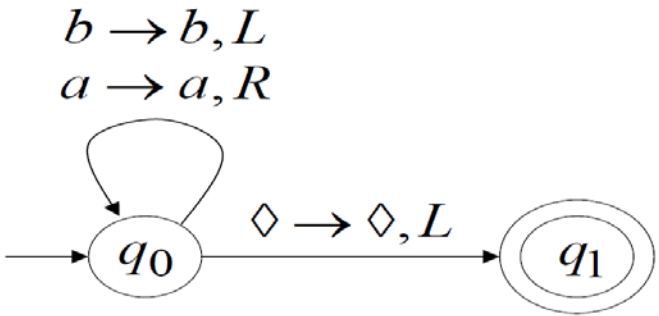
Input alphabet $\Sigma = \{a, b\}$

Accepts the language: a^*



Infinite Loop Example

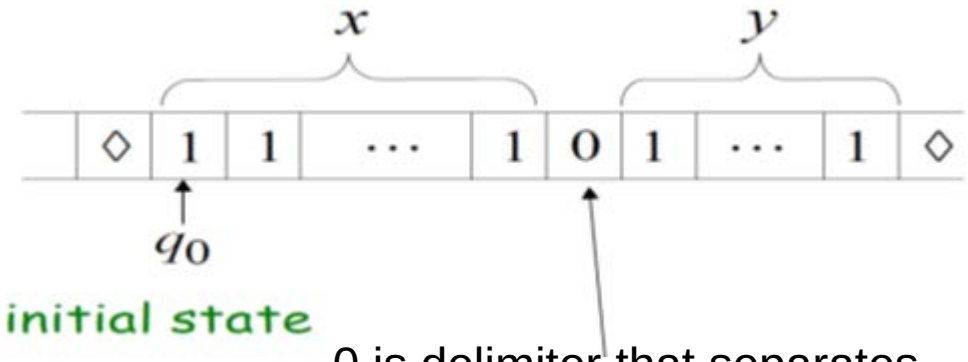
A Turing machine
for language $a^* + b(a+b)^*$



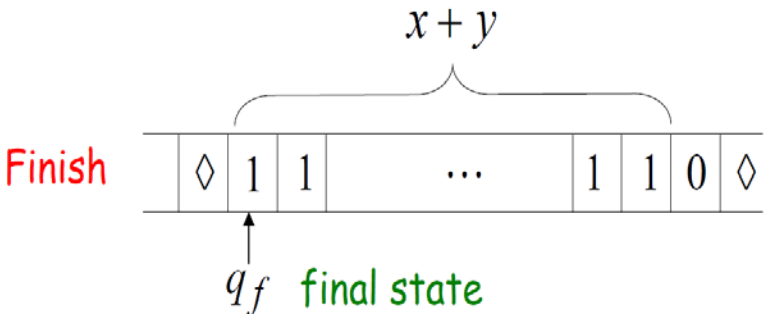
Computing Functions with TM

represent integer as unary is easier => 1 --> 1, 11 -> 2, 111 -> 3

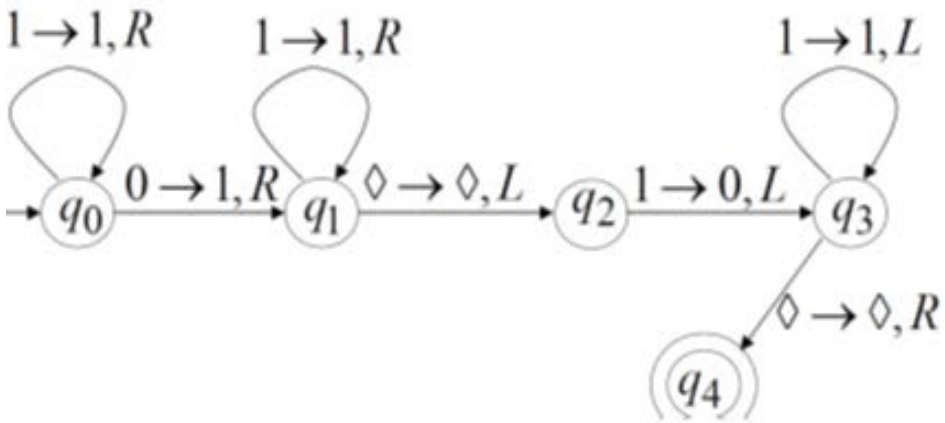
- The function f(x,y) = x+y is computable (x,y : integer)
- Turing Machine: Input string: x0y outputstring: xy0



0 is delimiter that separates the numbers

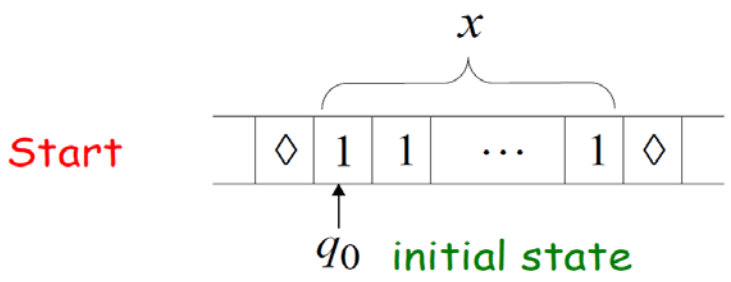


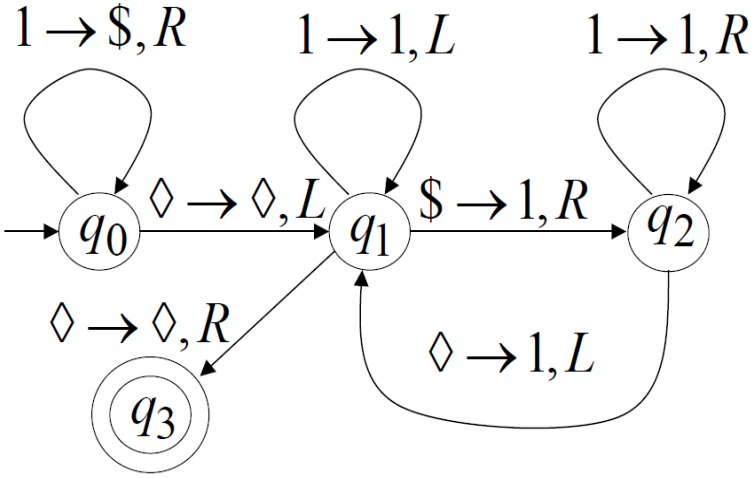
Ex) x=11, y=11 x+y= 1111(4)

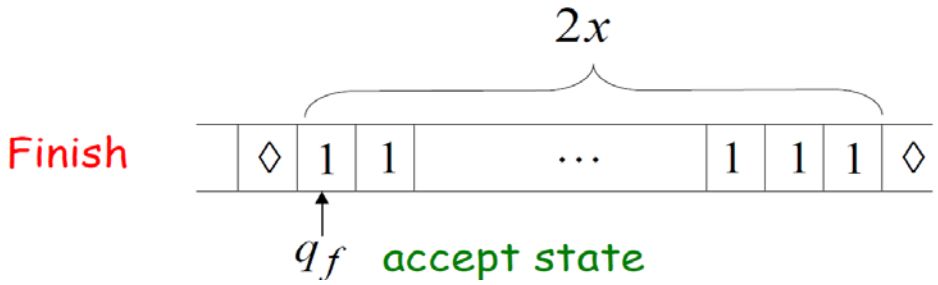


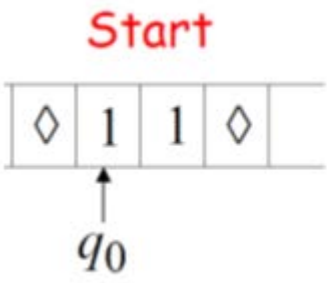
example

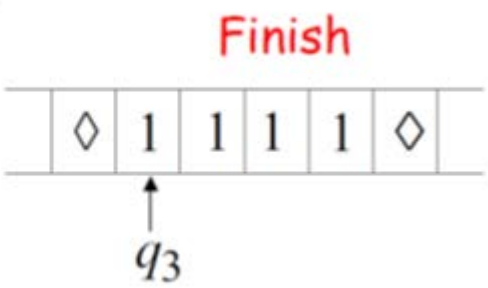
- f(X)= 2x (x: integer)
- TM: input string x (unary), output string: xx (unary)











example

- f(x,y) = 1, if x>y, Input: x0y output: 1 or 0
  0, if x≤y

Ex) “Δ 111 Δ 11Δ” → 1

- Pseudocode:

  Repeat

  Match 1 from x with 1 from y

  Until all of x or y is matched

  If 1 from x is not matched erase tape, write 1 (x>y)

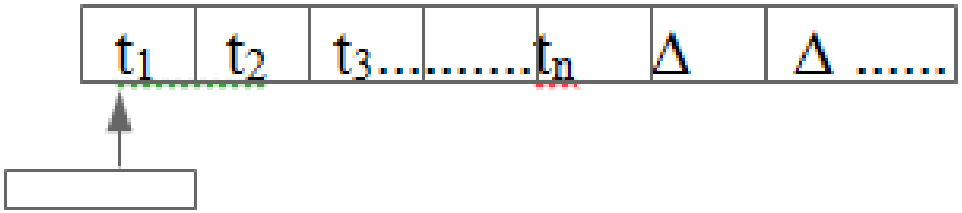
  else erase tape, write 0 (x≤y)

Instantaneous description (x, s, y)

* if word on tape form x.y then
  x: from beginning to before current
  y: current to the last

Let $x = t_1 t_2 \ldots t_{i-1}$, $y = t_i \ldots\ldots t_n$
(x, s, y) is ID S = state

Ex) Initial ID: (λ, So, y)



ex) Input "aaba" (state, symbol, move)

State	a	b	A	B	Δ
0	(1,A,R)	-	-	(3, B, R)	-
1	(1, a, R)	(2, B, L)	-	(1, B, R)	
2	(2, a, L)	-	(0, A, R)	(2, B, R)	-
3	-	-	-	(3, B, R)	(4, Δ,R)

Input alpha: {a, b} Tape alpha : {a, b, A, B, Δ}

Start State: So = 0 " -" : no next move

(sol) Initially input tape contains aabaΔΔΔ....

initial ID: (λ, 0, aaba)

(λ, 0, aaba) -> (A, 1, aba) -> (Aa, 1, ba) -> (A, 2, aBa) -> (λ, 2, AaBa) -> (A, 0, aBa) ->(AA, 1, Ba) -> (AAB, 1, a) -> (AABa, 1, λ) => Halt

<< TM as a Language Recognizer >>

ex) Design TM to recognize Language L consisting of words of the form $a^n b^n$, $n \geq 1$.

Then trace the evolution of ID's when the input is $a^2 b^2$.

(sol) A = {a,b,A,B, Δ} I = {a, b} O={A, B}

State	a	b	A	B	Δ
0	(1,A,R)	-	-	-	-
1	(1, a, R)	(1, b, R)	-	(2 B L)	(2, Δ, L)
2	-	(3, B, L)			
3	(3, a, L)	(3, b, L)	(4, A, R)	-	-
4	(1, A, R)	-	-	(5, B, R)	-

aabbΔ ->Aabb -> Aabb -> Aabb -> AabbΔ -> Aabb -> AabB ->AabB

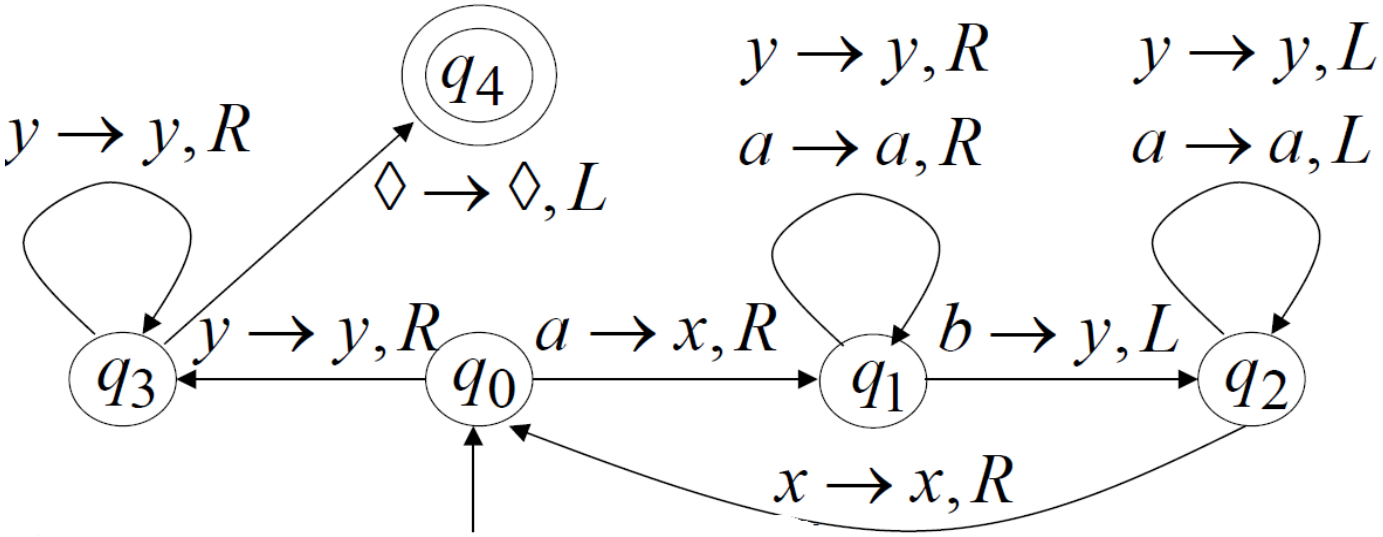
-> AabB -> AAbB -> AAbB ->AAbB -> AABB -> AABB

--> AABB --> AABBΔ

(λ, 0, aabb) -> (A, 1,abb) -> (Aa, 1, bb) -> (Aab, 1, b) -> (Aabb, 1, λ)

-> ,,,,

Turing machine for the language $\{a^n b^n\}$ $n \geq 1$



Pseudocode:

Match a's with b's

Repeat:

replace leftmost a with x

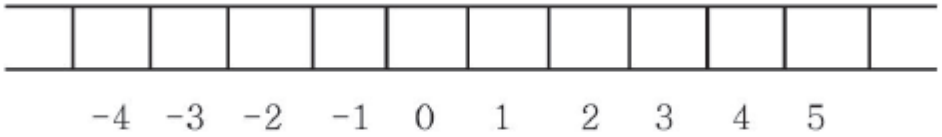
find leftmost b and replace it with y

Until there are no more a's or b's

If there is a remaining a or b then reject

Turing Machine의 종류

- Two-way TM



- Multiple TM