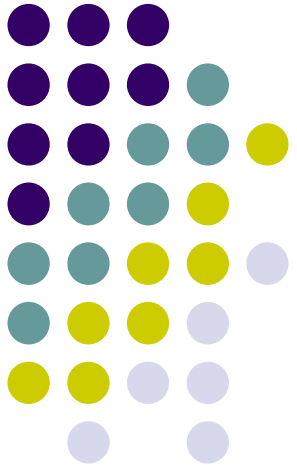
Finite Automata

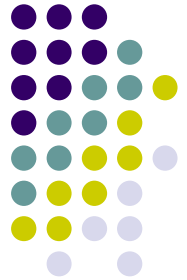
นำเสนอโดย
ดร.สุธี สุดประเสริฐ



คอมพิวเตอร์คืออะไร?

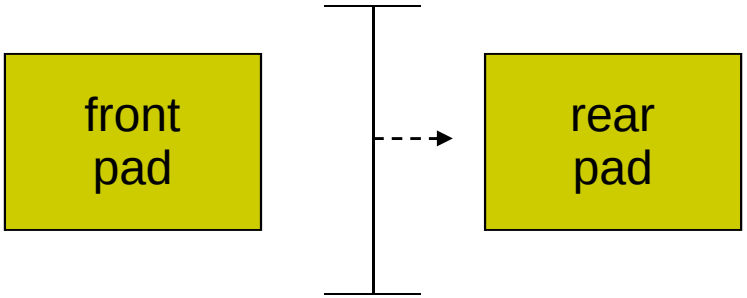
- ทฤษฎีการคำนวณเริ่มต้นจากปัญหาที่ว่า: คอมพิวเตอร์คืออะไร?
  - ดูเหมือนเป็นปัญหาไร้สาระหรือเปล่า?
- แต่คอมพิวเตอร์จริงๆมีความซับซ้อนมาก
  - ดังนั้นเราจะศึกษาคอมพิวเตอร์ในอุดมคติแทนซึ่งเรียกว่า *แบบจำลองทางการคำนวณ* (computational model).
- ในที่นี้เราจะเริ่มศึกษาแบบจำลองที่เรียบง่ายที่สุดก่อนซึ่งเรียกว่า **finite state machine** หรือ **finite automaton**.

Finite Automata



- finite automata เป็นแบบจำลองที่ดีสำหรับคอมพิวเตอร์ที่มีหน่วยความจำที่จำกัด
  - แล้วคอมพิวเตอร์ที่มีหน่วยความจำน้อยๆ (จำได้แค่สถานะปัจจุบันเท่านั้น) สามารถทำอะไรได้บ้าง?
- ตัวควบคุมอุปกรณ์ไฟฟ้า เช่น ประตูไฟฟ้าที่สามารถเปิด-ปิด อัตโนมัติเองได้ เป็นตัวอย่างของคอมพิวเตอร์ที่มีหน่วยความจำขนาดเล็ก

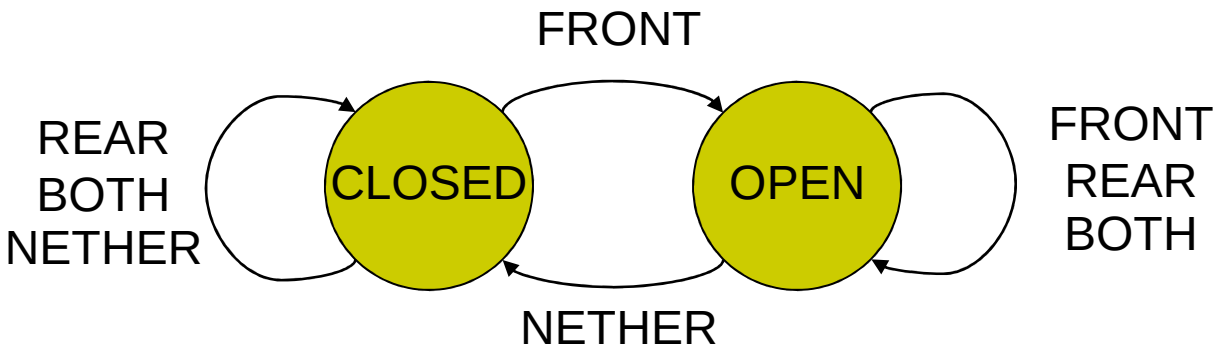
ตัวอย่าง (1)



ประตูแบบเหวี่ยง

- มี 2 สถานะคือ
  - OPEN และ CLOSED (เปิดประตู และ ปิดประตู)
- มีอินพุต 4 แบบ
  - FRONT, REAR, BOTH, และ NEITHER
  - คนอยู่ด้านหน้า คนอยู่ด้านหลัง คนอยู่ทั้งด้านหน้าและหลัง และ ไม่มีคนอยู่ทั้งหน้าและหลัง

ตัวอย่าง (2)

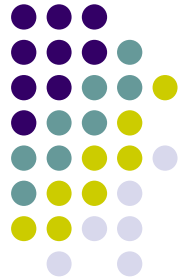


สัญญาณอินพุต

สถานะปัจจุบัน

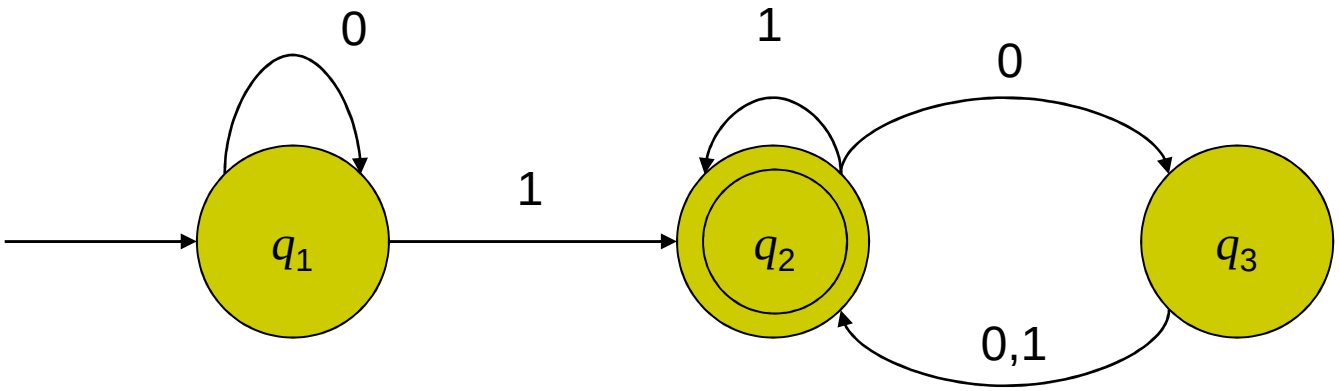
	NETHER	**FRONT**	**REAR**	**BOTH**
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

Finite Automata



- ก่อนที่เราจะอธิบาย finite automata โดยใช้คณิตศาสตร์ เราเริ่มศึกษาภาพรวมของ finite automata ในรูปแบบทั่วไปที่ไม่ได้เจาะจงสำหรับการประยุกต์ใช้ในงานใดงานหนึ่งก่อน

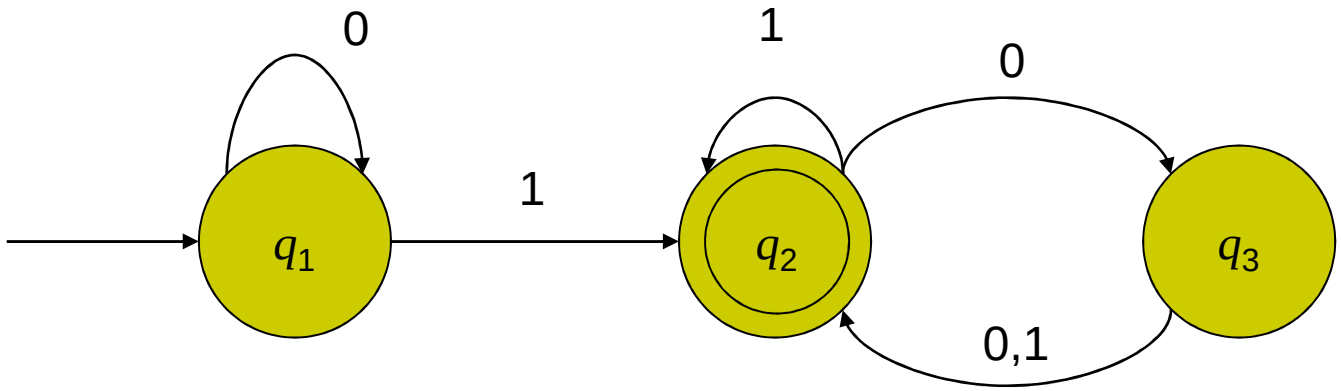
Finite Automata



A finite automaton called M_1 that has three *states*

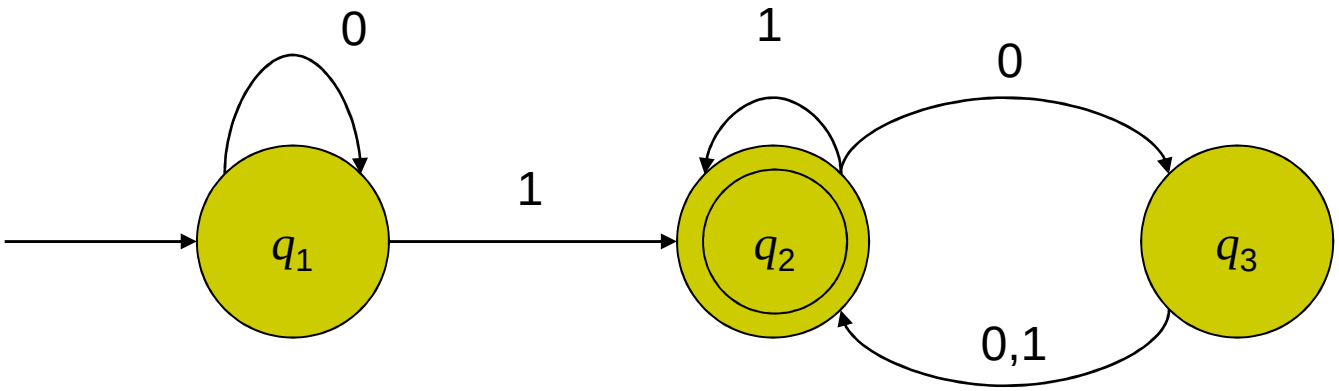
- The *start state* is q_1
- The *accept state* is q_2
- The arrows going from one state to another are called *transitions*
- The output is either *accept* or *reject*.

Finite Automata



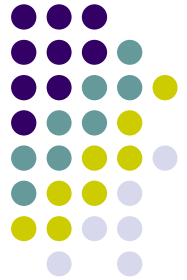
- เมื่ออินพุตคือ “1101”,
  1. Start in state q_1.
  2. Read 1, follow transition from q_1 to q_2.
  3. Read 1, follow transition from q_2 to q_2.
  4. Read 0, follow transition from q_2 to q_3.
  5. Read 1, follow transition from q_3 to q_2.
  6. Accept because M_1 is in an accept state q_2 at the end of the input.

Finite Automata



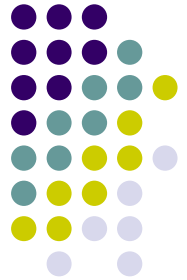
- ภาษาที่ประกอบด้วย strings ที่ M_1 ยอมรับ อธิบายได้ว่าอย่างไร?
  - 1,01,11, and 0101010101.
  - 100, 0100, 110000, and 0101000000.

Formal Definition



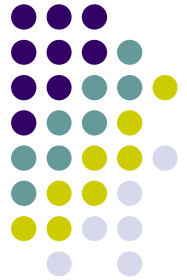
- แม้ว่าการใช้ state diagrams จะง่ายในการทำความเข้าใจ, เราก็ยังต้องการนิยามอย่างเป็นทางการ (formal definition) เพราะ
  - ประการแรก formal definition มีความชัดเจน
    - จะทำให้เราเห็นชัดเจนว่าอะไรที่ finite automata สามารถทำได้หรือไม่สามารถทำได้
  - ประการที่สอง formal definition ทำให้เราสามารถใช้สัญลักษณ์ (notation) แทนได้.
    - การใช้สัญลักษณ์ทำให้เราสามารถคิดและแสดงความคิดได้ชัดเจนขึ้น

Formal Definition



- formal definition สำหรับ finite automaton สามารถกำหนดได้โดยใช้วัตถุ 5 ตัวต่อไปนี้
  - set of state
  - input alphabet
  - rules for moving
  - start state
  - accept states

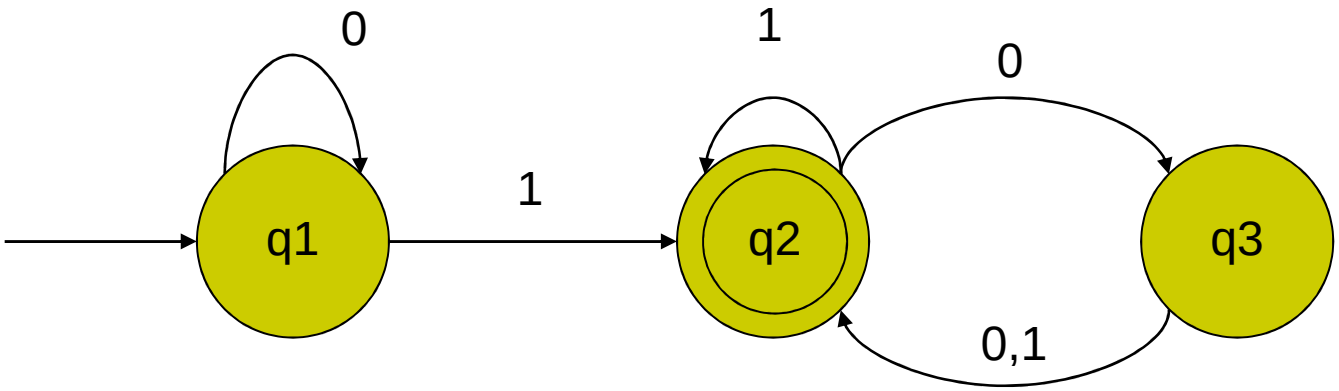
Formal Definition



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

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

Formal Definition



We can describe M_1 formally by writing $M_1 = (Q, \Sigma, \delta, q_1, F)$, where

- $Q = \{ q1, q2, q3 \}$
- $\Sigma = \{ 0, 1 \}$
- δ is described as

	0	1
q1	q1	q2
q2	q3	q2
q3	q2	q2

$\delta(q1,0) = q1$ $\quad \delta(q1,1) = q2$
$\delta(q2,0) = q3$ $\quad \delta(q2,1) = q2$
$\delta(q3,0) = q2$ $\quad \delta(q3,1) = q2$

- q1 is the start state, and
- $F = \{ q2 \}$.

Formal Definition

- ถ้า *A* คือเซตของสตริงทั้งหมดที่เครื่องจักร *M* ยอมรับ, เราพูดได้ว่า *A* คือภาษาของเครื่องจักร *M* และเขียนได้ว่า *L(M) = A*
- เราพูดว่า *M* **recognizes** *A* หรือ *M* **accepts** *A*.
- หนึ่งเครื่องจักรอาจจะยอมรับหลายสตริง, แต่สามารถจดจำได้แค่ภาษาเดียวเท่านั้น
  - สำหรับเครื่องจักรที่ไม่ยอมรับสตริงใดเลย, เราบอกว่าเครื่องจักรนั้นจดจำภาษาว่าง (empty language ∅)

Extended Transition Function

$\delta : Q \times \Sigma \rightarrow Q$ is the transition function

$\delta(q, a)$ = "the state to which the machine M goes if it is in state q and receives input symbol a"

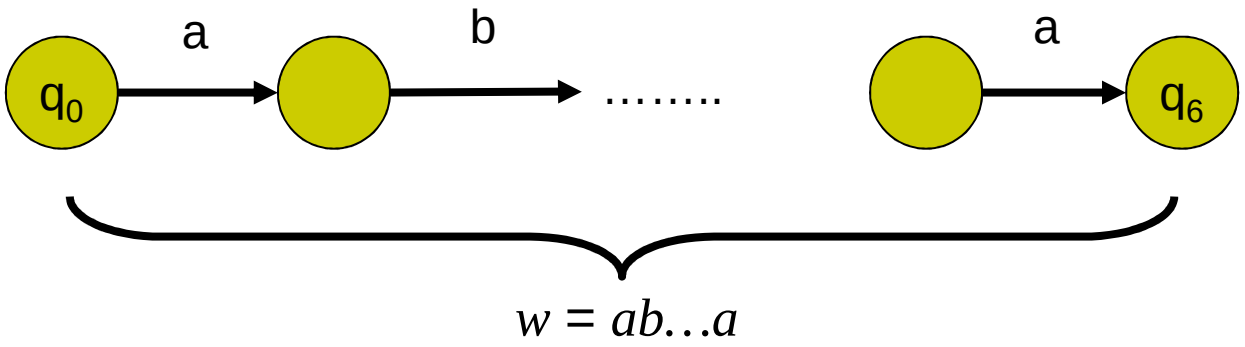
$\delta^*: Q \times \Sigma^* \rightarrow Q$ is the extended transition function

$\delta^*(q, w)$ = "the state in which M ends up, if it begin in state q, and receives the string w of several symbols"

Extended Transition Function

ตัวอย่าง:

$$\delta^*(q_0, w) = q_6$$



Extended Transition Function

Definition: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA.
Define the function

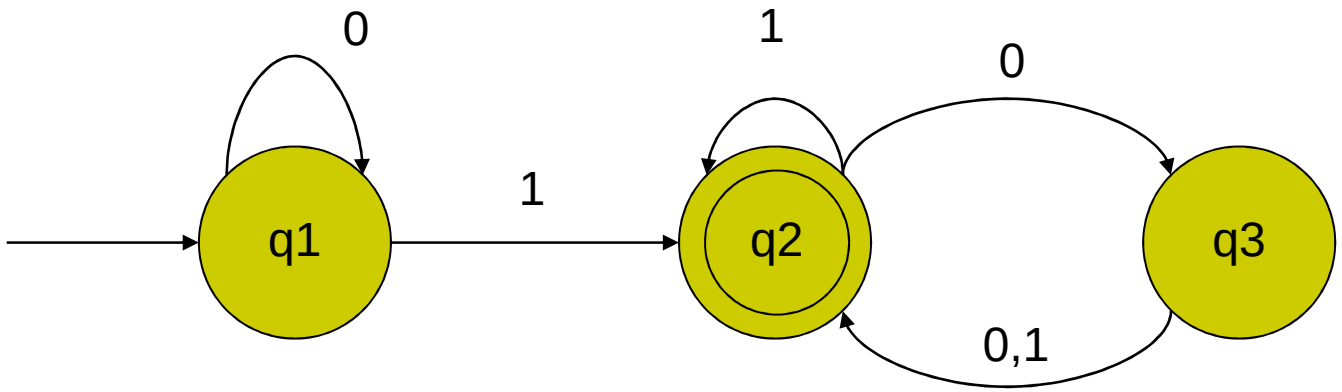
$$\delta^*: Q \times \Sigma^* \rightarrow Q$$

recursively as follows.

When $q \in Q$, $w \in \Sigma^*$, and $a \in \Sigma$,

1. $\delta^*(q, \varepsilon) = q$
2. $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$

Extended Transition Function



$\delta^*(q_1,010)$

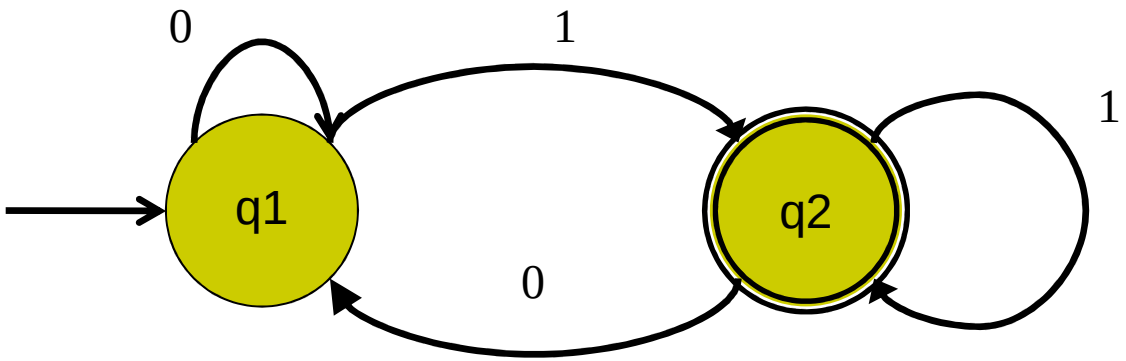
$\delta^*(q_1, 010) = \delta(\delta^*(q_1,01), 0)$

$\delta^*(q_1, 01) = \delta(\delta^*(q_1,0), 1)$

$\delta^*(q_1,0) = \delta(\delta^*(q_1,\varepsilon), 0)$

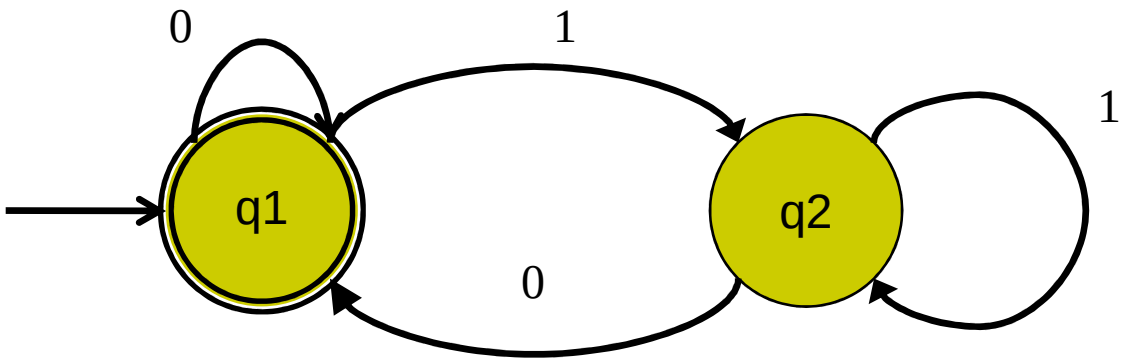
$\delta^*(q_1,\varepsilon) = q_1$

Example of Finite Automata (1)



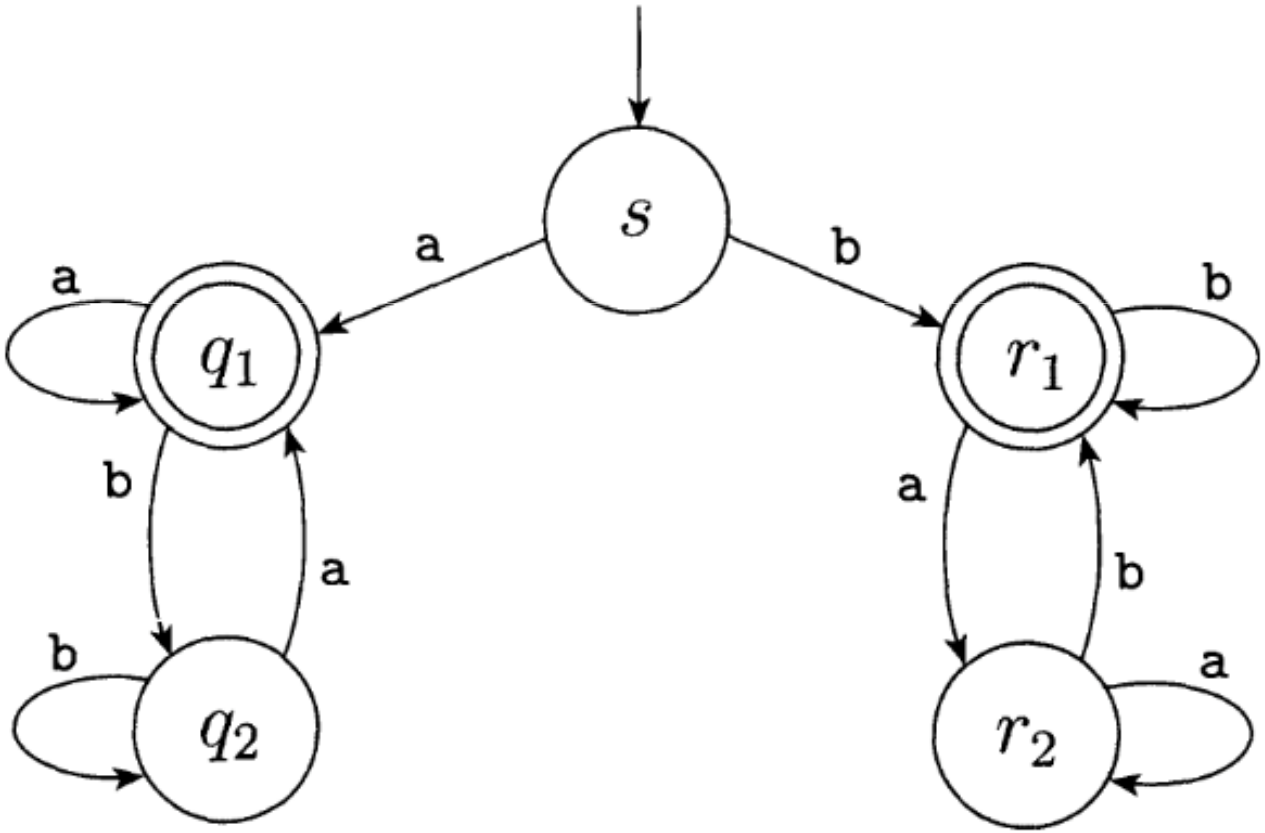
- Formal description คือ ?
- L(M) = ?

Example of Finite Automata (2)

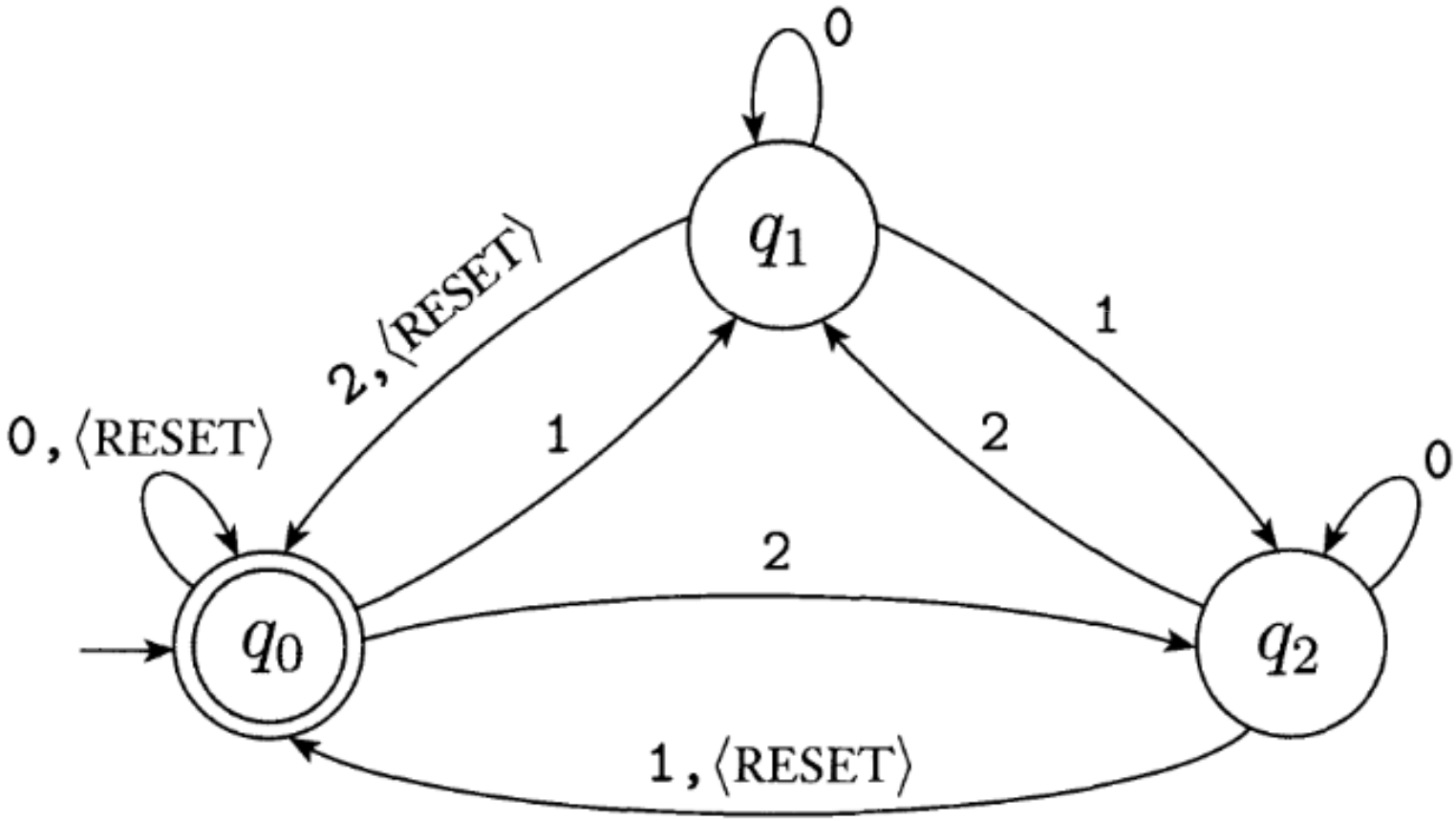


- Formal description คือ ?
- L(M) = ?

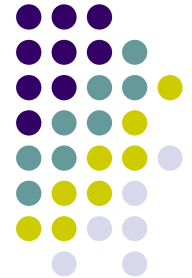
Example of Finite Automata (3)



Example of Finite Automata (4)



Formal Definition of Computation



- ตอนนี้เราทราบถึงวิธีการคำนวณอย่างคร่าวๆของ FA ต่อไปจะเป็นการนิยามอย่างเป็นทางการของคำนวณของ FA

Formal Definition of Computation

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton
- Let $w = w_1 w_2 \ldots w_n$ be a string where each w_i is a member of the alphabet Σ.
- Then M accepts w if $\delta^*(q_0, w) \in F$.
- We say that M recognizes language A if

  $A = \{ w \mid M \text{ accepts } w \}$
- A language is called a ***regular language*** if some finite automaton recognizes it.

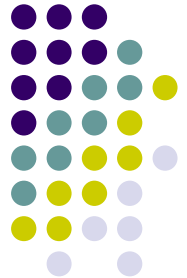
การออกแบบ Finite Automata

- การออกแบบ automata เหมือนงานศิลปะซึ่งต้องอาศัยความคิดสร้างสรรค์
- ซึ่งไม่มีสูตรสำเร็จตายตัว ☹
- อย่างไรก็ตามยังมีแนวทางที่เป็นประโยชน์ ในการนำไปใช้เพื่อออกแบบ automata ชนิดต่างๆ ได้ ☺
- หลักการคือให้คิดว่าตัวเองเป็น automata ที่เราต้องการออกแบบ หรือ เรียกว่า
  - ***"reader as automaton" method***

ตัวอย่าง (1)

- สมมุติให้ alphabet คือ {0, 1} ให้สร้าง FA ที่จดจำภาษาที่ประกอบด้วยสตริงทั้งหมดที่มี 1 เป็นเลขคี่
- เริ่มต้น เราต้องพิจารณาให้รู้ว่า อะไรคือสิ่งที่เราจำเป็นจะต้องจำในขณะที่เรากำลังอ่านข้อมูล
  - ทำไมเราจำข้อมูลที่ต้องการทั้งหมดเลย?
- ในกรณีนี้ สิ่งที่เราจำเป็นต้องจำคือจำนวนของ 1 ว่าเป็นจำนวนคู่หรือจำนวนคี่

ตัวอย่าง (2)



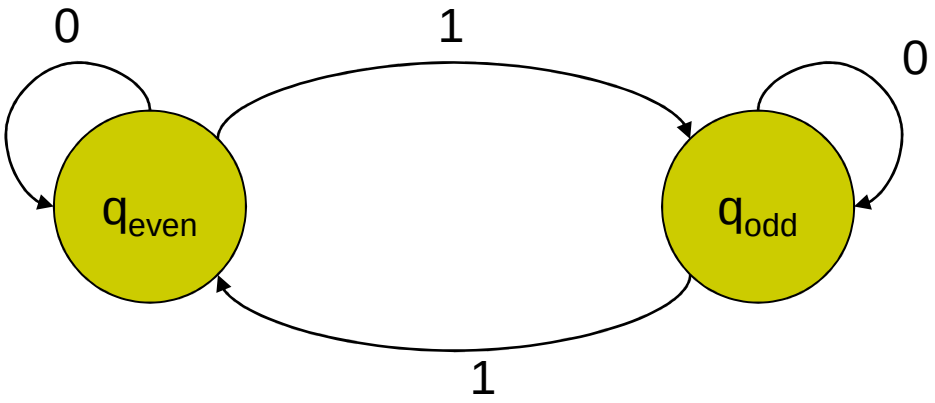
- จากนั้นจึงสร้างสถานะที่เป็นไปได้ทั้งหมด

ตัวอย่าง (3)

- ต่อไปจึงกำหนดการเปลี่ยนสถานะที่เป็นไปได้ทั้งหมด ซึ่งเกิดการอ่านอินพุตสตริง
- ในกรณีตัวอย่าง สถานะจะถูกเปลี่ยนก็ต่อเมื่ออินพุตเป็น 1 และจะอยู่กับที่เมื่ออินพุตเป็น 0

ตัวอย่าง (3)

- เราจะเขียนการเปลี่ยนสถานะได้ดังนี้

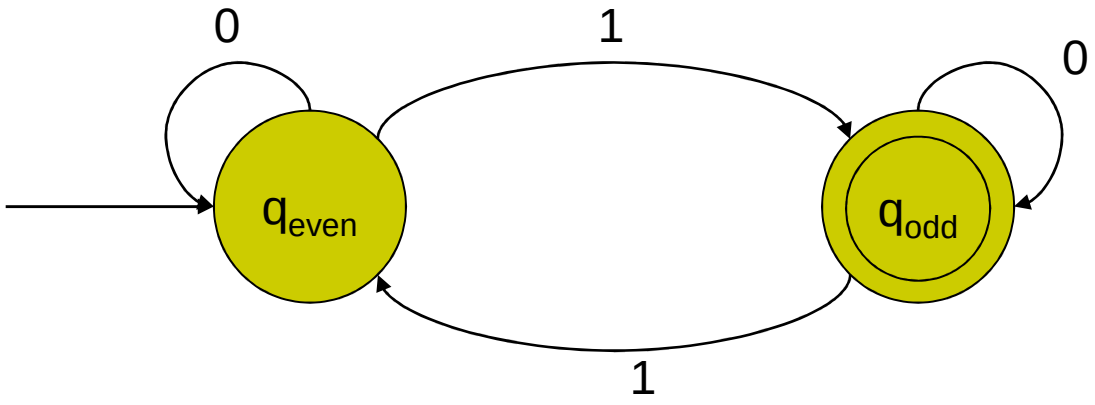


ตัวอย่าง (4)

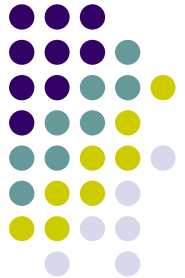
- ต่อไปจึงกำหนดสถานะเริ่มต้น ซึ่งก็คือสถานะที่ยังไม่มีอินพุตเข้ามาหรือสตริงขนาด 0 ตัว (สตริงว่าง ε)
- จากนั้นจึงกำหนดสถานะยอมรับ ให้สอดคล้องกับรูปแบบที่เราต้องการจะยอมรับจากอินพุตสตริง

ตัวอย่าง (5)

- เพิ่มสถานะเริ่มต้นและสถานะยอมรับ

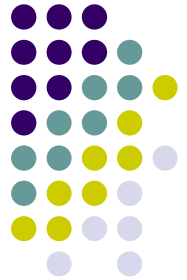


แบบฝึกหัด



- จงวาดแผนภาพของ DFA ที่สามารถจดจำภาษาต่อไปนี้ โดยที่ $\sum = \{0,1\}$
  - อักษรตัวแรกต้องแตกต่างจากตัวสุดท้าย
  - เมื่อมี 00 ปรากฏต้องมี 1 ตามมาเสมอ
  - สตริงทุกตัวต้องมี 00 แต่ต้องไม่มี 000

The Regular Operations



Definition

- Let A and B be languages. We define the regular operations *union*, *concatenation*, and *star* as follows.
- **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Concatenation**: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
- **Star**: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

ตัวอย่าง

- ให้อักษร (alphabet) Σ เป็นตัวหนังสือภาษาอังกฤษมาตราฐาน 26 ตัว {a, b,..., z}.
- ถ้า A = {good, bad} และ B = {boy, girl}, then

  $A \cup B$ = {good, bad, boy, girl}

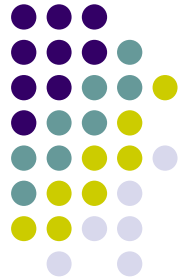
  $A \circ B$ = {goodboy, goodgirl, badboy, badgirl}

  A^* = {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, goodbadgood, goodbadbad, ...}

The Regular Operations

- กลุ่มของวัตถุใดจะมี **สถานะปิด** (closed) ภายใต้ตัวดำเนินหนึ่ง ถ้าตัวดำเนินนั้นถูกใช้กับสมาชิกของกลุ่มของวัตถุนั้นแล้วผลลัพธ์ที่ได้ยังคงอยู่ในกลุ่มของวัตถุเดิม
- กลุ่มของ regular languages มีสถานะปิดภายใต้ regular operations ทั้งสามตัว
- ต่อไปจะเป็นการพิสูจน์

The Regular Operations

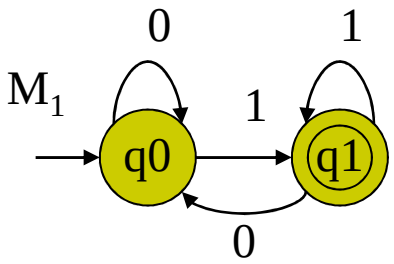


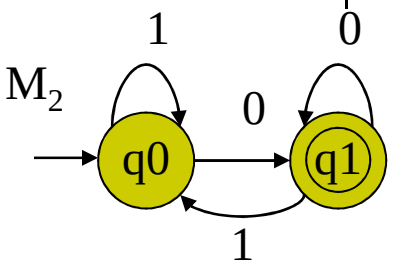
- **Theorem**
  - กลุ่มของ regular languages มีสถานะปิดภายใต้ตัวดำเนินการ union
  - ถ้า A_1 และ A_2 คือ regular languages, ดังนั้น $A_1 \cup A_2$ ก็เป็น regular languages ด้วย
- **Proof Idea**:
  - A_1 และ A_2 เป็น regular ดังนั้น
    - มี finite automaton M_1 ที่จดจำ A_1 ได้ และ
    - มี finite automaton M_2 ที่จดจำ A_2 ได้
  - เราจะแสดงว่า finite automaton M ซึ่งจดจำ $A_1 \cup A_2$ และ M ถูกสร้างจาก M_1 และ M_2

The Regular Operations

- **Proof Idea (ต่อ)**:
  - M ต้องยอมรับสตริงแบบเดียวกับที่ M_1 **หรือ** M_2 ยอมรับ
  - M จะทำงาน โดยการจำลองทำงานของ M_1 และ M_2 โดยที่จะยอมรับสตริงใด ก็ต่อเมื่อการจำลองนั้นยอมรับสตริงนั้น
  - เราจำเป็นต้องจำคู่ของสถานะทั้งหมดระหว่าง M_1 และ M_2 $(M_1 \times M_2)$
  - ดังนั้นการเปลี่ยนสถานะของ M จะเป็นการเปลี่ยนระหว่างคู่ ซึ่งเป็นการปรับเปลี่ยนสถานะของ M_1 และ M_2
  - สถานะยอมรับของ M คือคู่ที่ประกอบด้วยสถานะยอมรับของ M_1 หรือ M_2

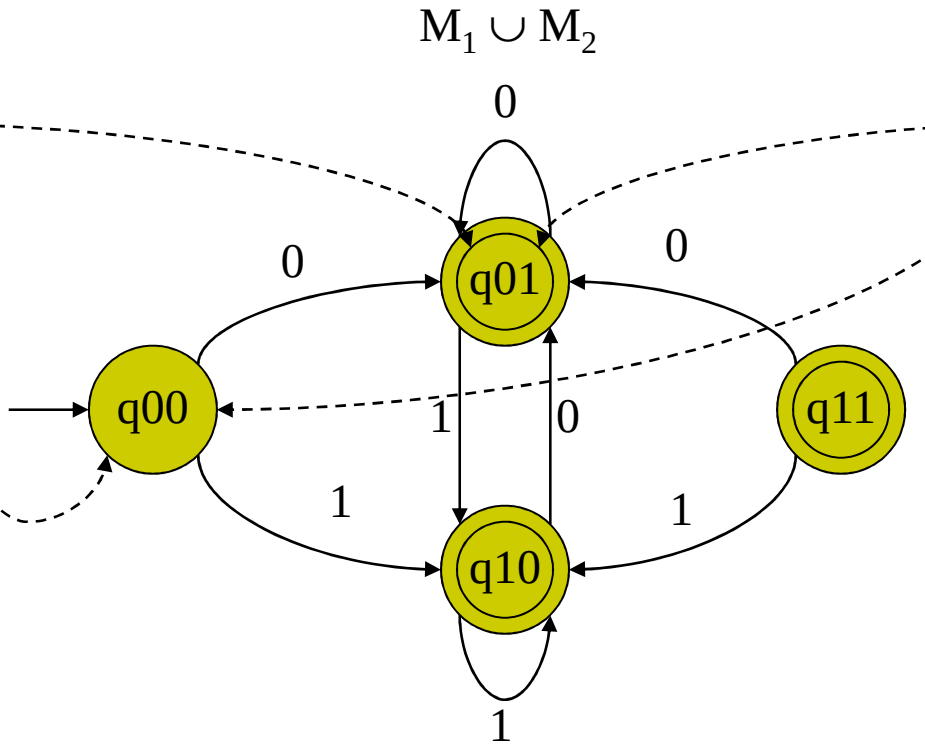
ตัวอย่าง



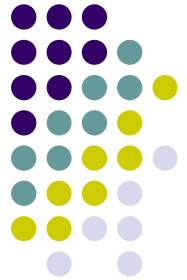


	0	1
q0	q0	q1
q1	q0	q1

	0	1
q0	q1	q0
q1	q1	q0



The Regular Operations



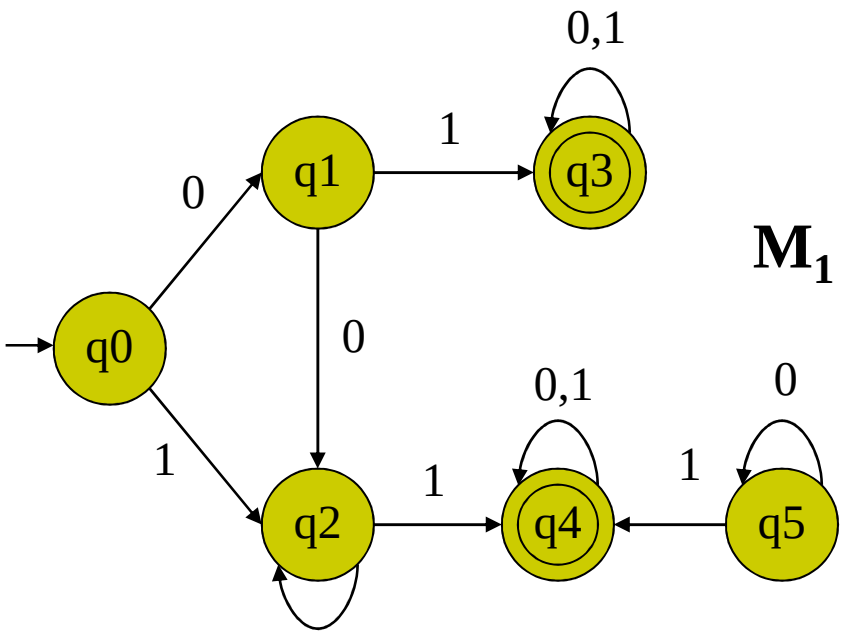
- **Theorem**
  - กลุ่มของ regular languages มีสถานะปิดภายใต้ตัวดำเนินการ concatenation
  - ถ้า A_1 และ A_2 เป็น regular languages ดังนั้น $A_1 \circ A_2$ ก็เป็น regular ด้วย
- **Proof Idea:**
  - ลองใช้แนวคิดแบบเดียวกับการพิสูจน์ของ union
  - เริ่มจาก finite automata M_1 และ M_2 ซึ่งจดจำ regular languages A_1 และ A_2.

The Regular Operations

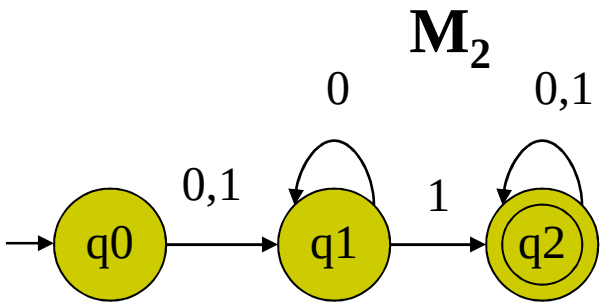
- **Proof Idea: (cont.)**
  - ในกรณีนี้ M จะยอมรับ ถ้าเราสามารถแยกอินพุตสตริงเป็น 2 ส่วนโดยที่ให้ M_1 ยอมรับส่วนแรกและให้ M_2 ยอมรับในส่วนที่สอง
  - แต่ปัญหาคือเราไม่รู้ว่าจะแบ่งอินพุตที่ตรงไหน ☹
  - เพื่อแก้ปัญหานี้เราจะศึกษาวิธีการใหม่ที่เรียกว่า nondeterminism.

การลดจำนวนสถานะใน FA

- FA ใดๆจะจดจำภาษาได้เพียงภาษาเดียว แต่หนึ่งภาษาสามารถถูกจดจำได้จากหลาย FA



M_1



M_2

M_1 และ M_2 จดจำภาษาเดียวกัน

การลดจำนวนสถานะใน FA

นิยาม:

สองสถานะ p และ q ของ FA ตัวหนึ่งจะถูกเรียกว่า ***indistinguishable***

ถ้า

$\delta^*(p, w) \in F$ implies $\delta^*(q, w) \in F$

และ

$\delta^*(p, w) \notin F$ implies $\delta^*(q, w) \notin F$

for all $w \in \Sigma^*$.

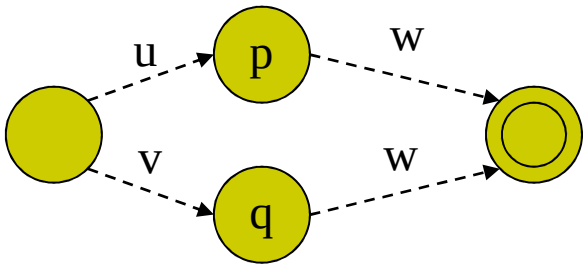
การลดจำนวนสถานะใน FA

ในทางกลับกัน ถ้ามีสตริง $w \in \Sigma^*$ ที่

$$\delta^*(p, w) \in F \text{ และ } \delta^*(q, w) \notin F,$$

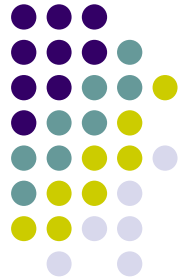
หรือกลับกัน เช่นนี้เราจะเรียกสถานะ p และ q ว่า ***distinguishable*** โดยสตริง w

ตัวอย่างของ indistinguishable



$\delta^*(p, w) \in F$ implies $\delta^*(q, w) \in F$

การลดจำนวนสถานะใน FA



Indistinguishability มีคุณสมบัติของ equivalence relations:

1. p R p
2. if p R q then q R p
3. if p R q then q R r then p R r

การลดจำนวนสถานะใน FA

วิธีการหนึ่งที่ใช้สำหรับลดจำนวนสถานะของ FA คือ วิธีการที่ใช้หลักการในการหาและรวมสถานะที่เป็นแบบ indistinguishable

วิธีการนี้จะเริ่มต้นจากการหาคู่ของสถานะที่เป็นแบบ distinguishable ก่อน

การลดจำนวนสถานะใน FA

Procedure: mark

1. ลบสถานะที่เข้าถึงไม่ได้ออกทั้งหมด
2. พิจารณาคู่ของสถานะทั้งหมด (p, q) ถ้า $p \in F$ และ $q \notin F$ หรือกลับกัน ให้ mark (p, q) ว่าเป็น distinguishable
3. วนซ้ำขั้นตอนต่อไปนี้จนกว่าจะไม่สามารถ mark คู่ของสถานะใหม่เพิ่มเติมได้

   สำหรับทุกๆ คู่ (p, q) และทุกๆ $a \in \Sigma$,

   คำนวณ $\delta(p, a) = p_a$ และ $\delta(q, a) = q_a$

   ถ้า (p_a, q_a) ถูก mark ว่าเป็น distinguishable

   ให้ mark (p, q) ว่าเป็น distinguishable.

การลดจำนวนสถานะใน FA

เราจะใช้ผลของ mark ในการแบ่งเซตของสถานะ Q ของ FA เป็นเซตย่อยที่ไม่ร่วมกัน (disjoint subsets)

ตัวอย่าง:

$Q = \{0,1,2,3,4,5,6,7\}$

unmarked pairs = (1,2), (1,4), (2,4), (3,5), (3,7), (5,7) (คู่ที่ไม่แตกต่างกัน)

disjoint subsets = {0}, {1,2,4}, {3,5,7}, {6}

การลดจำนวนสถานะใน FA

Procedure: reduce

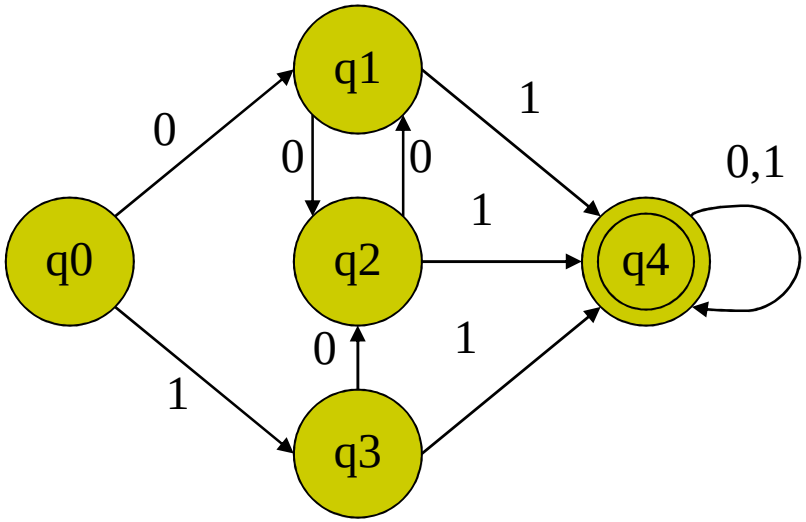
ถ้ามี FA $M = (Q, \Sigma, \delta, q_0, F)$, เราสามารถสร้าง FA ที่ถูกลดสถานะ $M' = (Q', \Sigma, \delta', q'_0, F')$ ได้ดังนี้

1. ใช้กระบวนการ mark เพื่อหาเซตของสถานะทั้งหมดที่ indistinguishable (ตามตัวอย่างที่แล้ว).
2. สร้างสถานะใหม่สำหรับ M' จากเซตของสถานะทั้งหมดที่ indistinguishable

   ตัวอย่างเช่น สร้าง q'_{124} จาก {1,2,4}.
3. เพิ่ม $\delta'(ij..k, a) = lm\ldots n$ ถ้า $q_r \in \{q_i, q_j, \ldots, q_k\}$ และ $q_p \in \{q_l, q_m, \ldots, q_n\}$ และ $\delta(q_r, a) = q_p$.
4. q'_0 สถานะเริ่มต้นของ M' คือสถานะที่ประกอบด้วย 0
5. F' คือเซตของสถานะที่ประกอบด้วย i โดยที่ $q_i \in F$.

การลดจำนวนสถานะใน FA

ตัวอย่าง



mark คู่ระหว่างสถานะ q4 กับสถานะอื่นๆ

1: (q0,q4), (q1,q4), (q2,q4), (q3,q4)

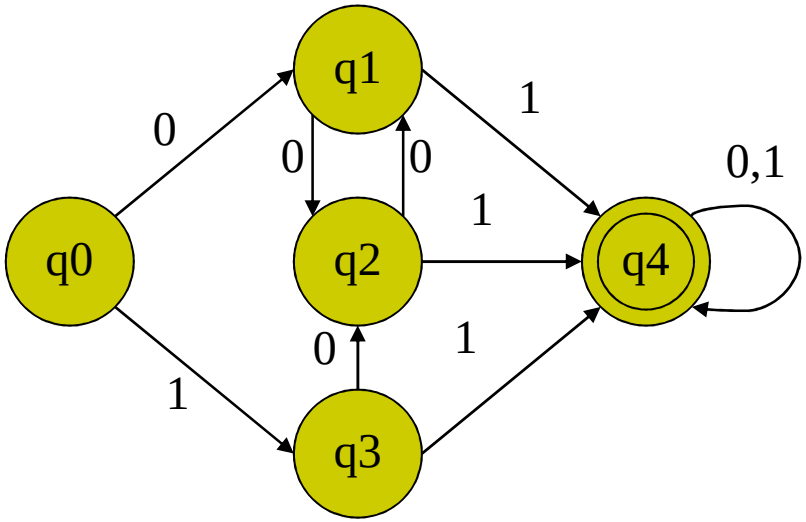
j

q1				
q2				
q3				
q4	1	1	1	1
	q0	q1	q2	q3

i

การลดจำนวนสถานะใน FA

ตัวอย่าง



	0	1
q0	**q1**	q3
q1	q2	q4
q2	**q1**	q4
q3	q2	q4
q4	**q4**	q4

marked
(q0,q4)
(**q1,q4**)
(q2,q4)
(q3,q4)

→ (q0,q4), (q2,q4)

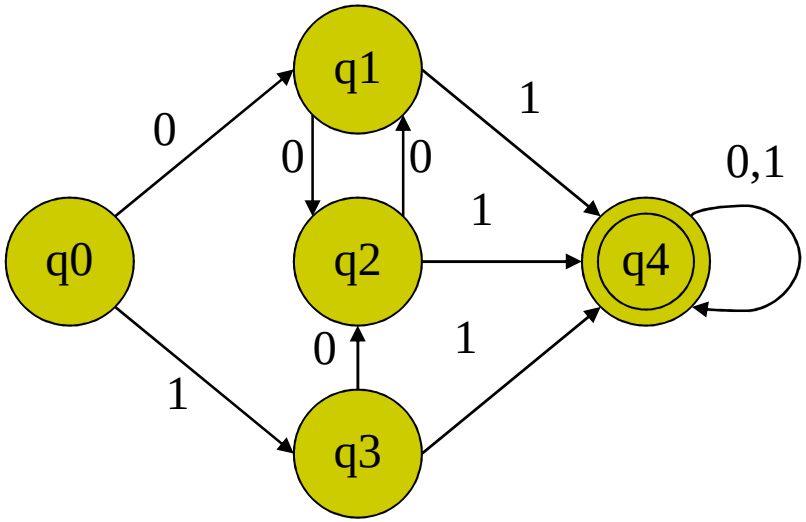
j

q1				
q2				
q3				
q4	1	1	1	1
	q0	q1	q2	q3

i

การลดจำนวนสถานะใน FA

ตัวอย่าง



	0	1
q0	q1	q3
q1	**q2**	q4
q2	q1	q4
q3	**q2**	q4
q4	**q4**	q4

marked
(q0,q4)
(q1,q4)
(**q2,q4**)
(q3,q4)

→ (q1,q4), (q3,q4)

2: (q0,q1), (q0,q2), (q0,q3)

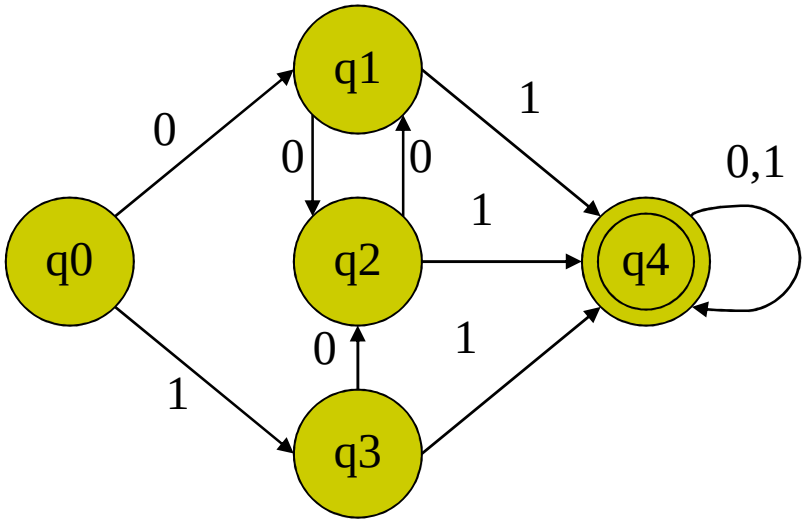
j

q1				
q2				
q3				
q4	1	1	1	1
	q0	q1	q2	q3

i

การลดจำนวนสถานะใน FA

ตัวอย่าง



	0	1
q0	q1	**q3**
q1	q2	**q4**
q2	q1	**q4**
q3	q2	**q4**
q4	q4	**q4**

marked
(q0,q4)
(q1,q4)
(q2,q4)
(**q3**,**q4**)

→ (q0,q1), (q0,q2), (q0,q3), (q0,q4)

2: (q0,q1), (q0,q2), (q0,q3)

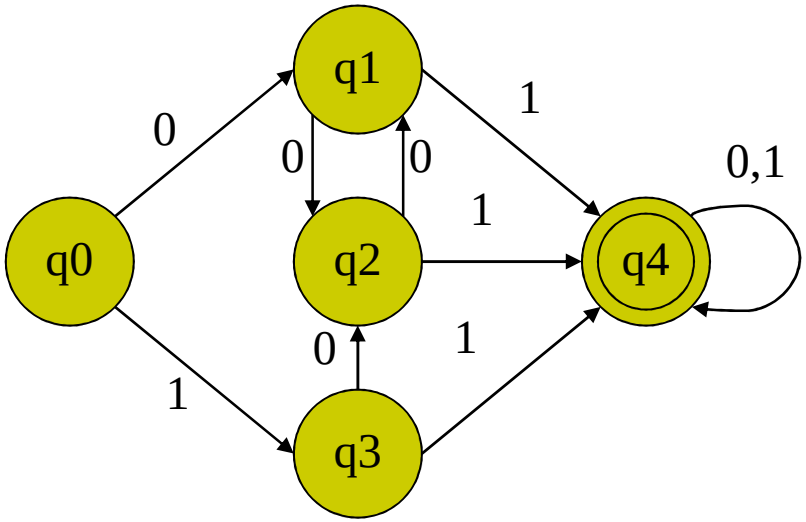
j

q1	2			
q2	2			
q3	2			
q4	1	1	1	1
	q0	q1	q2	q3

i

การลดจำนวนสถานะใน FA

ตัวอย่าง



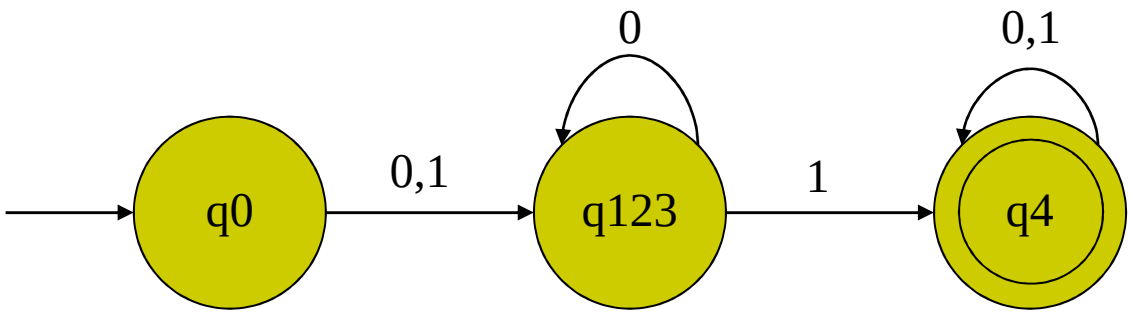
	0	1
q0	q1	q3
q1	q2	q4
q2	q1	q4
q3	q2	q4
q4	q4	q4

q1	2			
q2	2	x		
q3	2	x	x	
q4	1	1	1	1
	q0	q1	q2	q3

Unmarked pairs = (q1,q2), (q1,q3), (q2,q3)

การลดจำนวนสถานะใน FA

Q = {q0, q1, q2, q3, q4}
Unmarked pairs = (q1,q2), (q1,q3), (q2,q3)
The disjoint subsets = {q0}, {q1,q2,q3}, {q4}



	0	1
q0	q1	q3
q1	q2	q4
q2	q1	q4
q3	q2	q4
q4	q4	q4