204753

Theory of computation

Lecture 4: Regular expression

ในทางคณิตศาสตร์ เราได้ใช้ operation พวก + - ในการสร้างนิพจน์ทางคณิตศาสตร์ เช่น

$$(9 + 3) – 7$$

เช่นเดียวกัน เราจะใช้ regular operation ในการสร้างนิพจน์ที่อธิบายภาษา ซึ่งเราเรียกว่า regular expression

Regular expression

Regular expression สามารถนำมาใช้ในการอธิบายภาษาที่ซับซ้อนขึ้นได้จากการสร้างขึ้นมาจากก้อนเล็กๆ ด้วย regular operation เช่นเรารู้จักภาษากลุ่มหนึ่ง เราก็จับมา union กันได้

ตัวอย่างเช่น

$$(\{\mathbf{0}\} \cup \{\mathbf{1}\}) \circ \{\mathbf{0}\}^*$$

หรือเขียนแบบย่อเป็น $(0 \cup 1)0^*$

ขึ้นต้นด้วยอะไรก็ได้ลงท้ายด้วย 0 หรือไม่ลงก็ได้

นี่เรียกว่า regular expression เขียน 0 แทน {0}, 1 แทน {1}

รับอะไร

$$(0 \cup 1)^*$$

Definition

R เป็น **regular expression** ถ้า R มี

1. a for some a $\in \Sigma$
2. ε
3. $\emptyset$
4. $(R_1 \cup R_2)$ เมื่อ R_1 และ R_2 เป็น regular expression
5. $(R_1 \circ R_2)$ เมื่อ R_1 และ R_2 เป็น regular expression
6. (R_1^*) เมื่อ R_1 เป็น regular expression

Precedence

เมื่อเรามีตัวดำเนินการหลายตัว ถ้าเกิดเจอพร้อมกัน ทำตัวไหนก่อน

Operations ต่างๆ จะทำงานด้วยลำดับ

- $*$
- $\circ$
- $\cup$

การเขียนแบบย่อ

RR^*

RR^* สามารถเขียนได้เป็น R^+

การเขียนแบบย่อ

$RRRR$

$RRRR$ สามารถเขียนได้เป็น R^4 โดยทั่วไป R^k เป็น concatenation ของ R กับตัวมันเอง k ครั้ง

ตัวอย่างของ regular expression

1. 0^*10^*
2. $\Sigma^*1\Sigma^*$
3. $\Sigma^*001\Sigma^*$
4. $(01^+)^*$
5. $(\Sigma\Sigma)^*$
6. $01 \cup 10$
7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$
8. $1^*\emptyset$
9. $\emptyset^*$

คุณสมบัติ

- $R \cup \emptyset$
  - $R \cup \emptyset = R$
- $R \circ \varepsilon$
  - $R \circ \varepsilon = R$
- $R \cup \varepsilon$
  - $R \cup \varepsilon$ อาจจะไม่เท่ากับ R ขึ้นกับว่า R มี ε หรือไม่มี
- $R \circ \emptyset$
  - $R \circ \emptyset$ อาจจะไม่เท่ากับ R ขึ้นกับว่า R เป็น $\emptyset$ หรือไม่มี

Equivalence

A language is regular iff some regular expression describes it.

Iff ย่อมาจาก if and only if

A language is regular iff some regular expression describes it.

การพิสูจน์ว่าทฤษฎีข้างต้นเป็นจริงเนื่องจากเป็น iff ก็ต้องพิสูจน์สองทาง

1. If a language is described by a regular expression, then it is regular.
2. If a language is regular, then it can be described by a regular expression.

แบบฝึกหัด

กำหนดให้ $\Sigma = \{0,1\}$

จงหา FA M_1 ที่ recognize 01^+

จงหา FA M_2 ที่ recognize $(10)^*$

จงหา FA M_3 ที่ recognize $(01^+) \cup (10)^*$

จงหา FA M_4 ที่ recognize $1^+ \circ ((01^+) \cup (10)^*)$

A regular expression describes a regular language

If a language is described by a regular expression, then it is regular.

นั่นหมายความว่าเอา regular expression เราสามารถสร้าง finite automaton บางตัวที่ recognize ภาษานั้นได้

ในการพิสูจน์ เราจะมองดูว่า regular expression ถูกสร้างมาได้อย่างไร

Definition

R เป็น **regular expression** ถ้า R มี

1. a for some a $\in \Sigma$
2. ε
3. $\emptyset$
4. $(R_1 \cup R_2)$ เมื่อ R_1 และ R_2 เป็น regular expression
5. $(R_1 \circ R_2)$ เมื่อ R_1 และ R_2 เป็น regular expression
6. (R_1^*) เมื่อ R_1 เป็น regular expression

สังเกตว่าในการสร้าง regular expression มีวิธีในการสร้าง 6 วิธี

แต่ไม่ว่าจะสร้างด้วยวิธีไหนก็ตาม จะมี finite automaton ที่ recognize มันได้อยู่ดี

Rule 1

R is a regular expression if R is a for some $a \in \Sigma$

What is a DFA that recognizes R?

Rule 2

R is a regular expression if R is ε

What is a DFA that recognizes R?

Rule 3

R is a regular expression if R is $\emptyset$

What is a DFA that recognizes R?

Rule 4

R is a regular expression if R is $(R_1 \cup R_2)$ เมื่อ R_1 และ R_2 เป็น regular expression

กำหนด NFA N_1 และ N_2 ที่ recognize R_1 และ R_2 แล้ว NFA ที่ recognize R คืออะไร

Rule 5

R is a regular expression if R is $(R_1 \circ R_2)$ เมื่อ R_1 และ R_2 เป็น regular expression

กำหนด NFA N_1 และ N_2 ที่ recognize R_1 และ R_2 แล้ว NFA ที่ recognize R คืออะไร

Rule 6

R is a regular expression if R is (R_1^*) เมื่อ R_1 เป็น regular expression

กำหนด NFA N_1 ที่ recognize R_1 แล้ว NFA ที่ recognize R คืออะไร

- สังเกตว่า เราเริ่มจากหน่วยเล็กๆ (Base case, 3 case แรก) แล้วก็ค่อยๆ ทำ เช่นค่อยๆ union ค่อยๆ concate กัน

- ในการพิสูจน์ เราจะแสดงว่า NFA ของ regular expression เมื่อกำหนด NFAs ของ subexpression มาให้

- สังเกตว่ามันเป็น induction!!!

Inductive proof

ในบางครั้ง inductive proofs จะเรียกว่า structural induction

ปกติจำนวนเต็ม มี i แล้วก็มี i+1

แต่ทีนี้เรามี R_1 R_2 แล้วเรามี $R_1 \cup R_2, R_1 \circ R_2, R_1^*$

Inductive hypothesis: (เมื่อพิจารณา expression R)

assume ว่าสำหรับทุกๆ smaller regular expression R', R' สามารถอธิบายได้ด้วย NFA N'

แล้วก็สร้างโดย assume ว่ามี M_1 M_2 ทีนี้ก็สร้างออกมาได้

แบบฝึกหัด

จงหา NFA ที่ recognize $(01 \cup 0)^*$

ถ้าทำได้แล้วลองทำเป็น DFA

แบบฝึกหัด

จงหา NFA ที่ recognize $(0 \cup 1)^*010$

A language is regular iff some regular expression describes it.

การพิสูจน์ว่าทฤษฎีข้างต้นเป็นจริงเนื่องจากเป็น iff ก็ต้องพิสูจน์สองทาง

1. If a language is described by a regular expression, then it is regular. (ตรงนี้ทำไปแล้ว โดยพิจารณาว่า regular expression ถูกสร้างมาได้อย่างไร)
2. If a language is regular, then it can be described by a regular expression.(เดียวต่อไป)

Any regular expression can be described with a regular expression

ทำอย่างไรดี

มันมีอะไรให้เราบ้าง

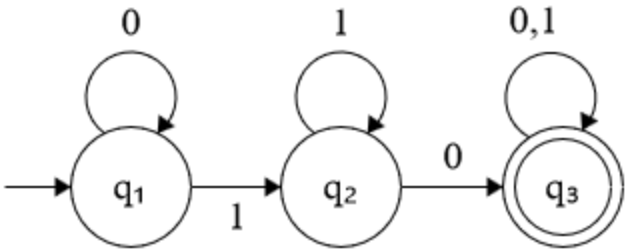
A is a regular language

แล้วหมายความว่าอะไร (ดูนิยาม)

มี DFA ที่ recognize A

M_1

Regular expression ที่อธิบายภาษาที่ recognize ด้วย M_1คืออะไร



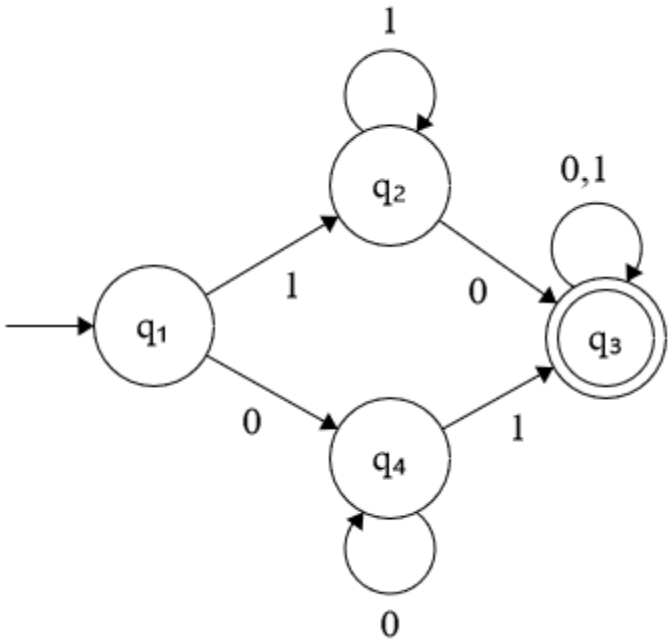
ลองแปลงมือดู

- อะไรที่เป็น loop ก็ทำเป็น *
- อะไรเป็นเส้นเชื่อมก็ใส่ต่อกัน

0*11*0(0U1)*

M_2

Regular expression ที่อธิบายภาษาที่ recognize ด้วย M_2คืออะไร



ขณะที่ลองหา regular expression ให้ลองคิดวิธีสำหรับสร้าง DFA

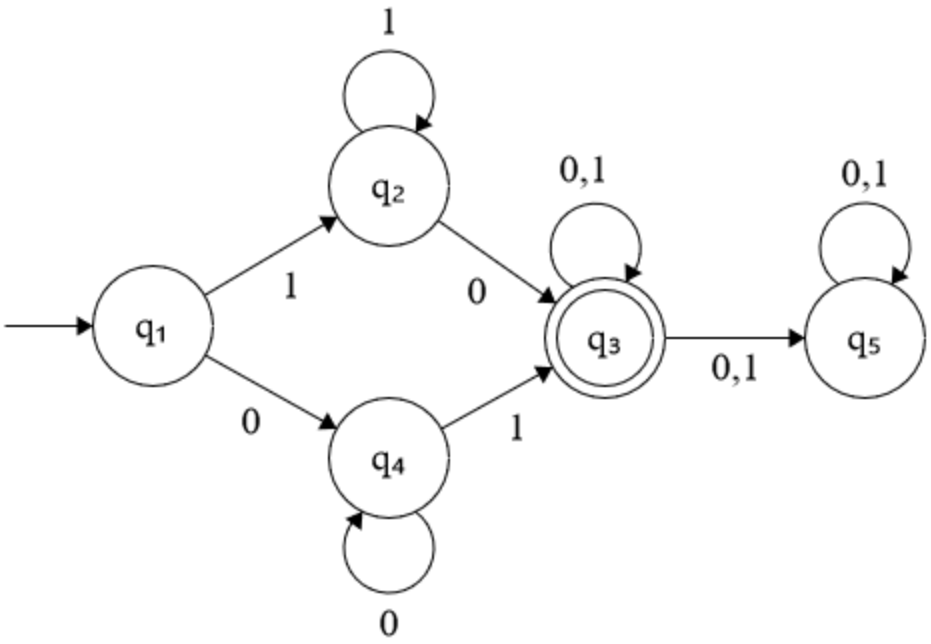
- คิดเป็นเส้นทางเดียวก่อน
- 11*0(0U1)* กับ 00*1(0U1)*

มีสองทางก็ union

- (11*0 U 00*1)(0U1)*

M_3

Regular expression ที่อธิบายภาษาที่ recognize ด้วย M_3คืออะไร

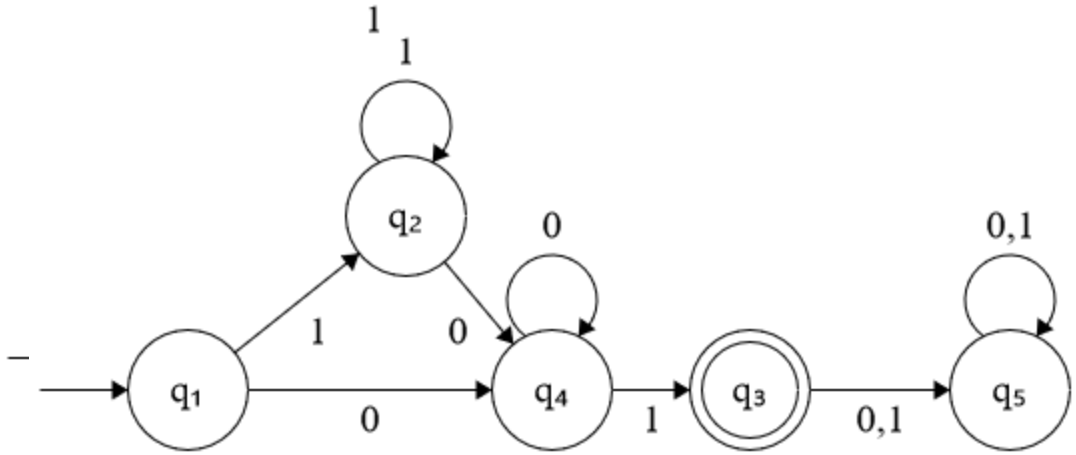


ขณะที่ลองหา regular expression ให้ลองคิดวิธีสำหรับสร้าง DFA

- ข้างหลังต้องไปไหม
- เราก็หาทางที่จะเดินไปให้ถึง accept state ก็พอ
- ให้ลองนึกด้วยว่าถ้าจะเขียนในรูปแบบทั่วไปจะเขียนอย่างไร

M_4

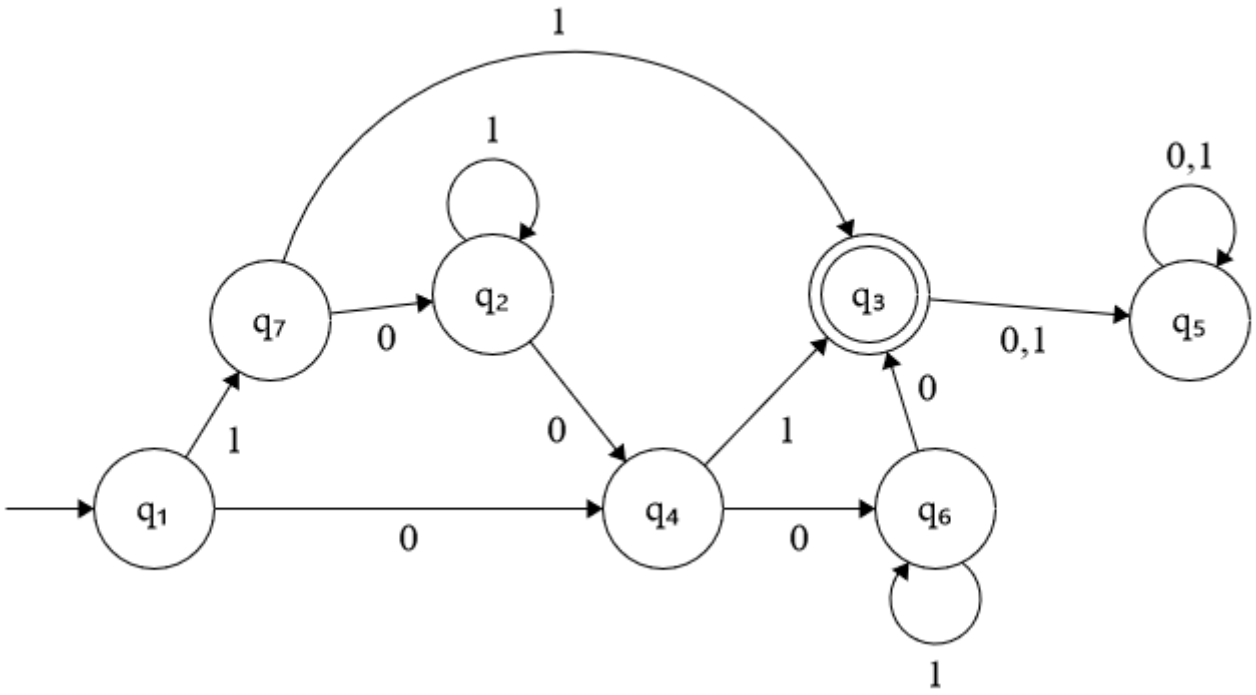
Regular expression ที่อธิบายภาษาที่ recognize ด้วย M_4คืออะไร



ขณะที่ลองหา regular expression ให้ลองคิดวิธีสำหรับสร้าง DFA

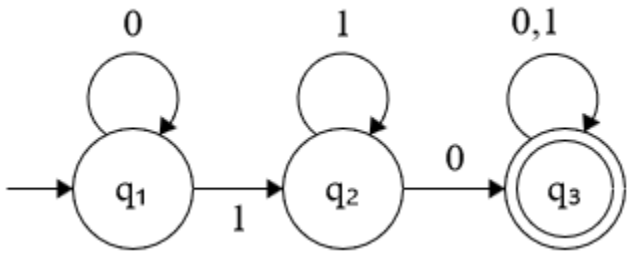
M_5

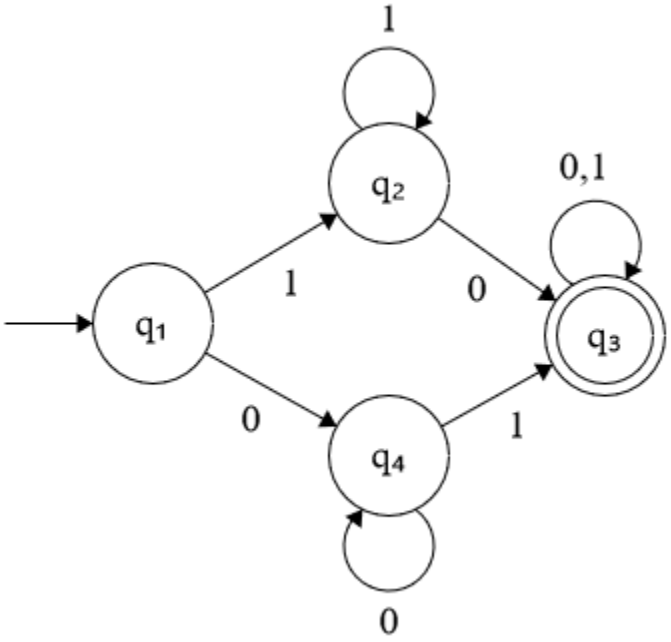
Regular expression ที่อธิบายภาษาที่ recognize ด้วย M_5คืออะไร



ขณะที่ลองหา regular expression ให้ลองคิดวิธีสำหรับสร้าง DFA

- พอเราใช้ DFA เราก็ต้องหาทางจัดการ
- นึกถึงว่าเราถ้าเราแปลงทั้งก้อนไม่ได้
- เราก็ทำให้มันเป็นก้อนเล็กๆ ที่อยู่ในรูปที่จัดการได้





- แต่พอเป็น M_5 มันก็ยุ่งยากมาก
- แทนที่เราจะแบ่งเป็นก้อนๆ เราจะทำอย่างอื่นแทน

- เราจะรู้ว่าทำอะไรยังไง เราก็ต้องรู้ก่อนว่าเราจะทำไปทำไม

- ถ้าจะค่อยๆ ทำก็ต้องบอกได้ว่าทำแล้วมันคืบหน้า

แทนที่เราจะแปลง DFA ทั้งหมดไปเป็น regular expression ในทีเดียว เราอาจจะต้องค่อยๆ ทำทีละนิด

ถ้าเราสามารถทำแล้วมีความคืบหน้าเรื่อยๆ จะทำให้เรามั่นใจได้ว่าจะทำมันเสร็จได้ **คล้ายกับ induction**

แล้วจะคืบหน้าได้อย่างไร

อาจจะถามแทนว่าเป้าหมายแบบไหนที่เราต้องการ

เป้าหมายเรา

Finite Automaton ที่ง่ายท่าสุดสำหรับการสร้าง regular expression

DFA ที่มี 2 states: start state กับ accept state หากทำได้แบบนี้ regular expression ของเราคืออะไร

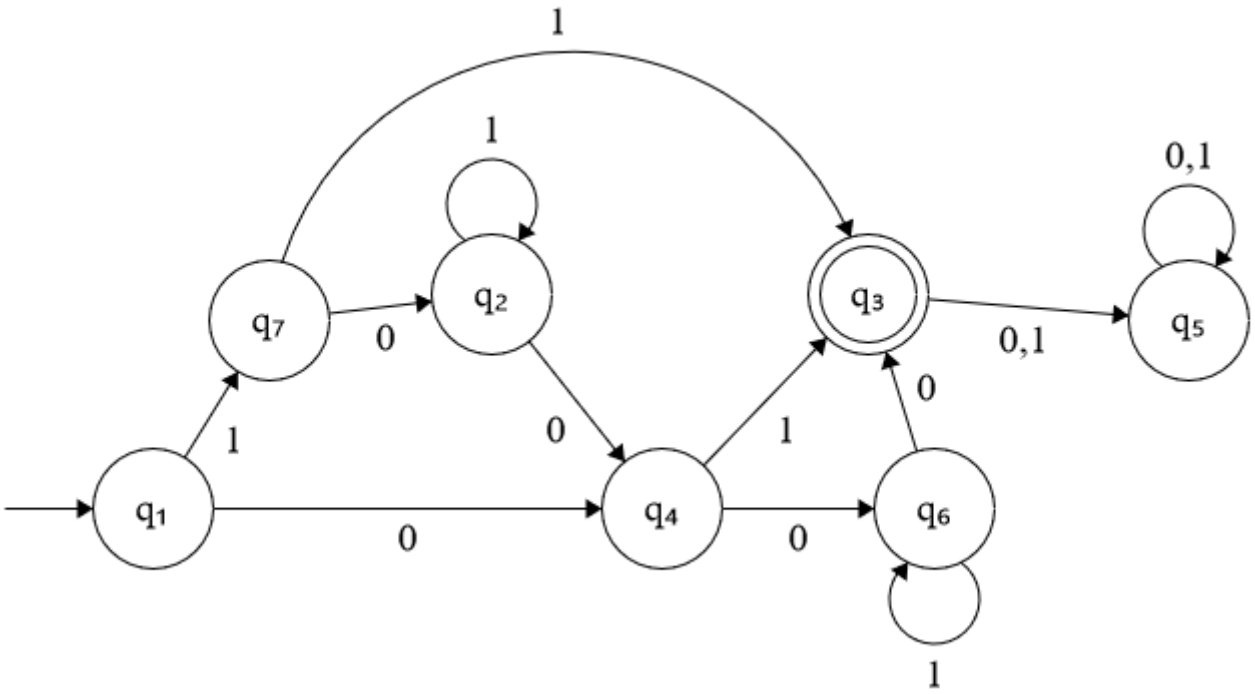
ของที่อยู่บนเส้นเชื่อม!!!

แล้วเราจะทำอย่างไร

เราพยายามที่จะลด state

แต่ละ step ก็ลดทีละ 1 state

จัดการกับ M_5

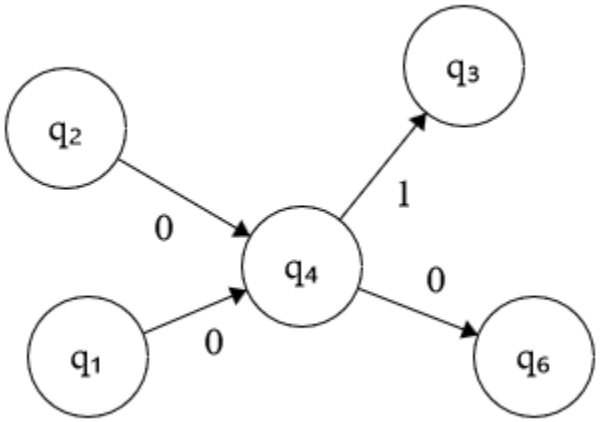


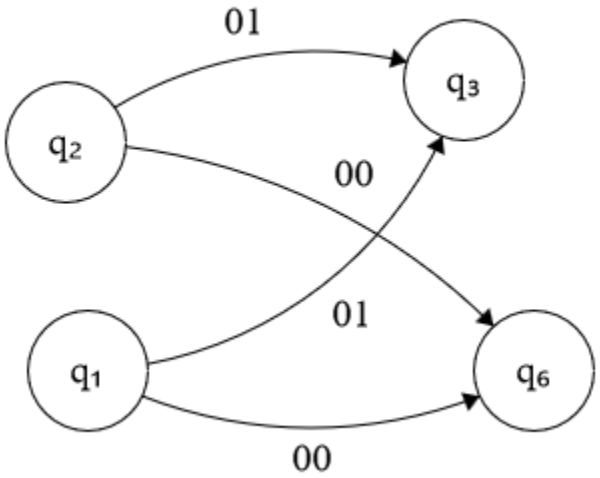
ลองลบ q_7 อาจจะเพิ่มเส้นเป็น 10 11

ลองลบ q_4

จะเอา q_4 ออก

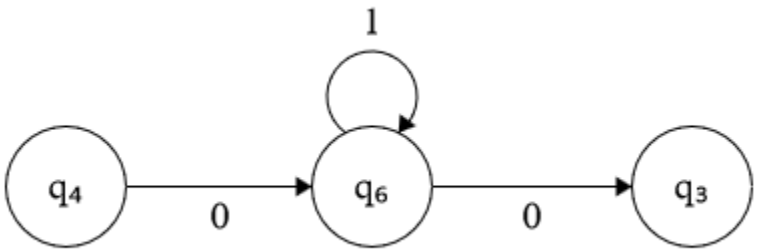
จะเอาออกได้ ก็ต้องดูว่า มีอะไรเข้าบ้าง ทุกๆ คู่ที่เข้าทุกๆ คู่ที่ออกต้องจับคู่กันได้

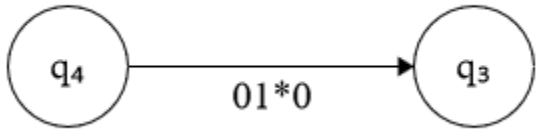




แปลงแล้วไม่ใช่ DFA บนเส้นเชื่อมนี้ก็ไม่ใช่ string มันมี * ได้

จัดการกับ q_6





เราต้องการกระบวนการการลด state เราอนุญาตให้ transition edge เป็น regular expression

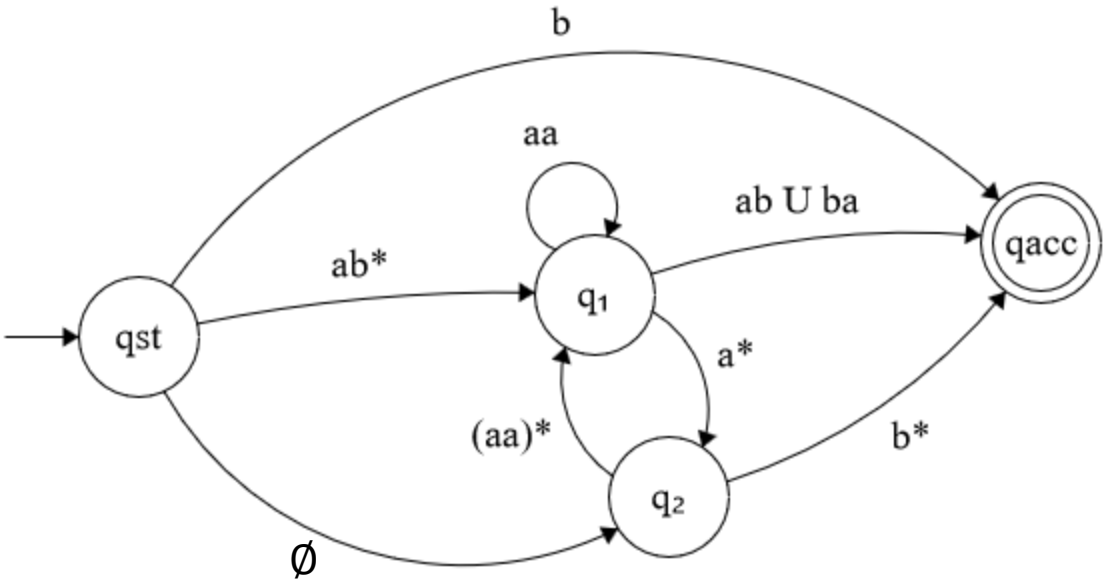
มันไม่ใช่ DFA NFA แล้ว เราจะนิยามว่าเป็น

generalized nondeterministic finite automata

Generalized nondeterministic finite automata คือ nondeterministic finite automata ที่เราอนุญาตให้ regular expression เป็น label ของลูกศร transition ได้

GNFA มันสามารถย้ายไปยัง state ใหม่ได้ถ้า block ของ input symbols ที่รับมานั้น match กับ regular expression บนลูกศรนั้น

ตัวอย่าง GNFA



Special form

เราต้องการ GNFA ที่มีคุณสมบัติบางอย่าง (รูปแบบนี้เปลี่ยนเป็น regular expression ได้ง่าย)

- Start state มีลูกศรชี้ไปยังทุก state แต่ไม่มีลูกศรชี้มาที่ start state จาก state อื่นๆ
- มี accept state อันเดียว และทุก state มีลูกศรชี้มาที่มัน แต่มันไม่ชี้ไปที่อื่น และ aceept state ไม่เป็น start state
- นอกจาก start และ accept state ทุก state จะมีลูกศรชี้หากันและมีลูกศรชี้ตัวเองด้วย

แนวทางที่เราจะทำ

เนื่องจาก language A เป็น regular เราจะมี DFA M ที่ recognize A

จาก M เราจะเปลี่ยนเป็น GNFA G (Part 1)

ถ้า G มีมากกว่า 2 states จะหา GNFA G' ที่มี state ที่น้อยลง (Part 2)

Part 1 DFA-> GNFA

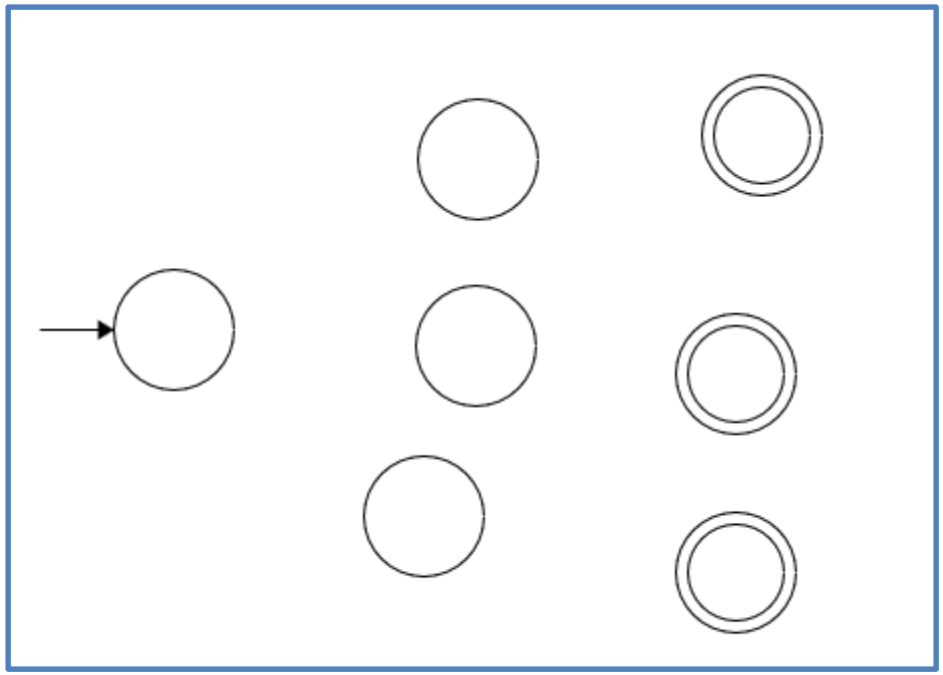
เมื่อกำหนด DFA มาให้ เราจะสร้าง GNFA G

เริ่มต้นเราจะเพิ่ม start state และ accept state ก่อน

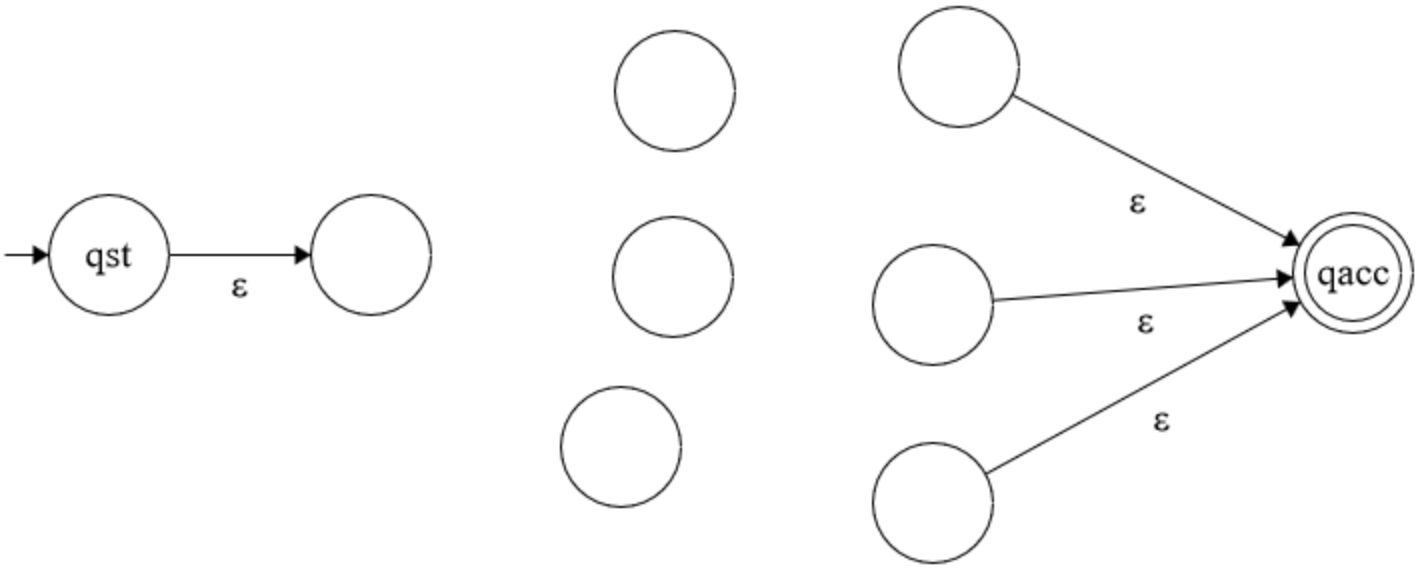
เรามีตัวช่วยคือ

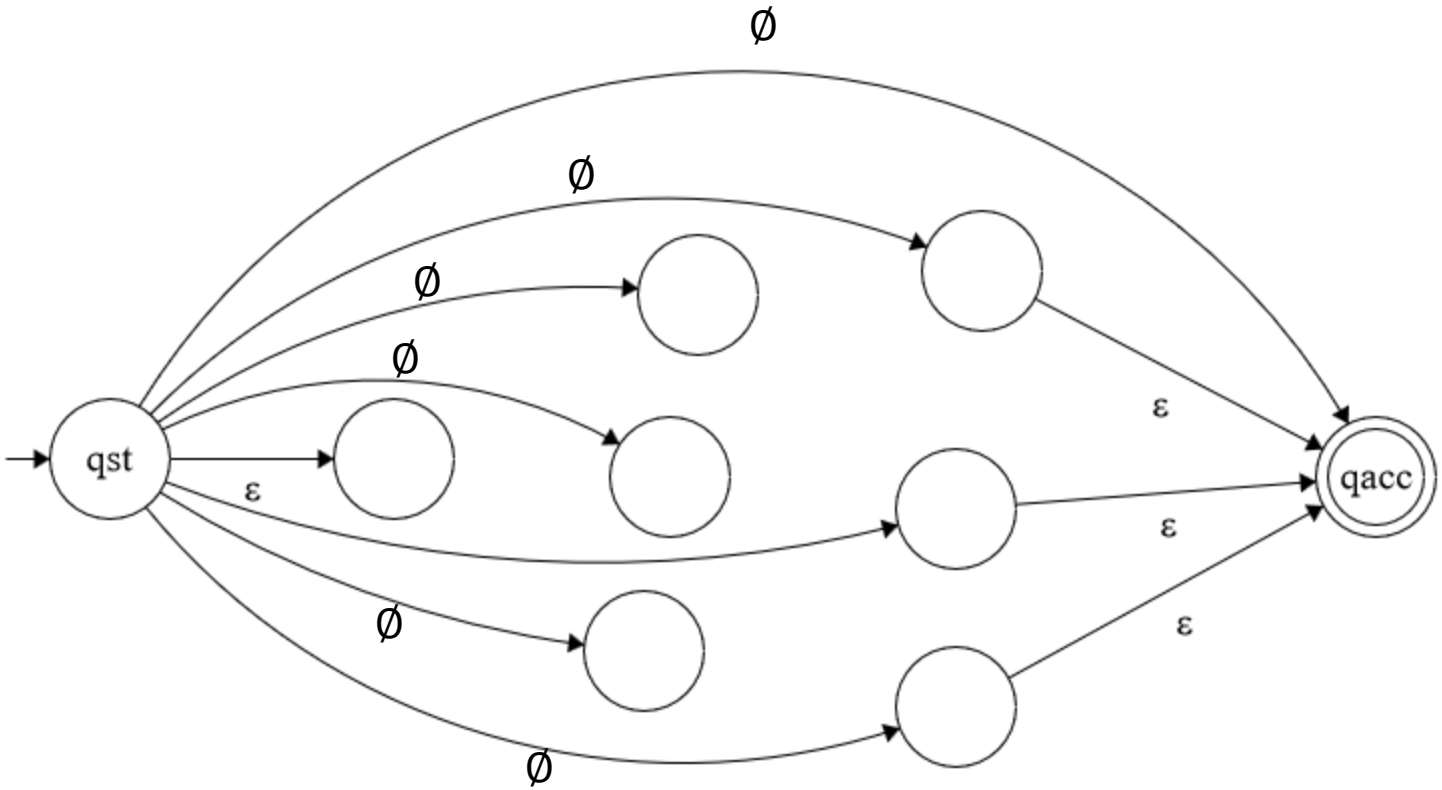
ลูกศรที่เป็น $\emptyset$

ลูกศรที่เป็น ε



- เติม start กับ accept ใหม่





- เพิ่ม Ø จากทุกโหนดไป accept และ ทุกโหนดวิ่งหากัน
- จาก Ø ถือว่าไม่รับ Ø concatอะไร หายหมด

DFA->GNFA การสร้าง

กำหนดให้ $M = (Q, \Sigma, \delta, q_0, F)$

- เพิ่ม start state ใหม่ q_{start} และเพิ่มลูกศรจาก q_{start} ไป q_0
- เพิ่ม accept state ใหม่ q_{accept} และเพิ่มลูกศรจาก ทุก state $q \in F$ ไปยัง q_{accept}
- เพิ่มลูกศรที่เหลือ label ด้วย Ø

Part 2 GNFA-> Regular expression

ถ้า GNFA G มี 2 state ง่าย เปลี่ยนได้ตรงๆ เลย

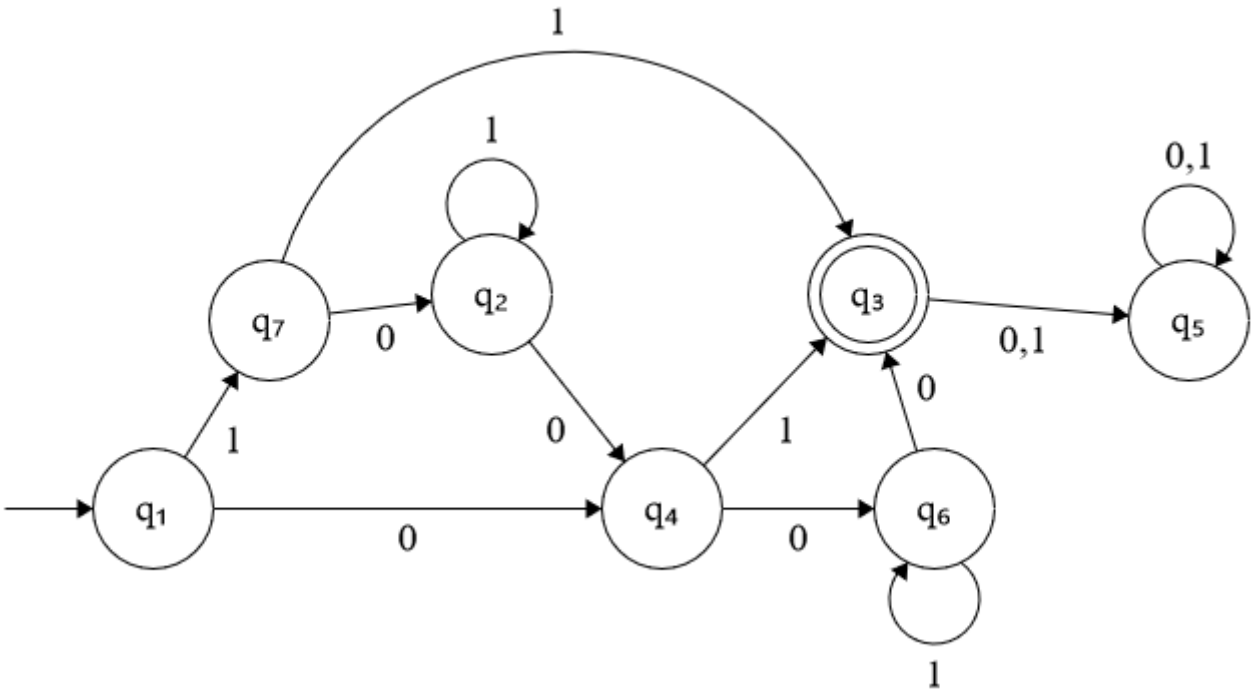
ถ้า GNFA G มีมากกว่า 2 states:

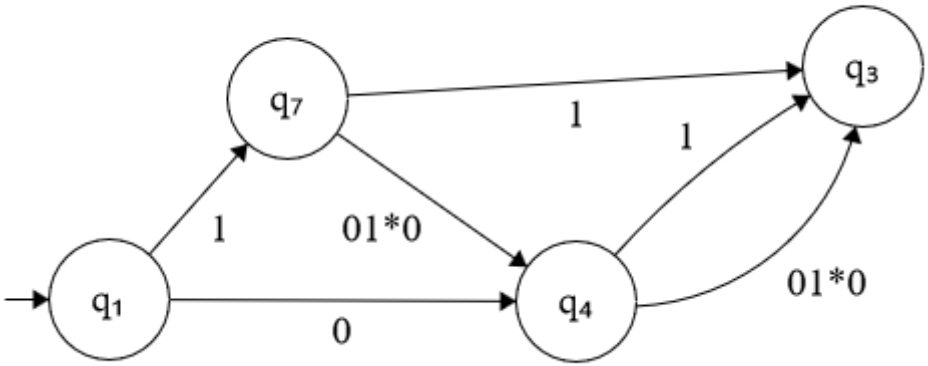
เลือกมา 1 state $q_{rip} \notin \{q_{start}, q_{accept}\}$

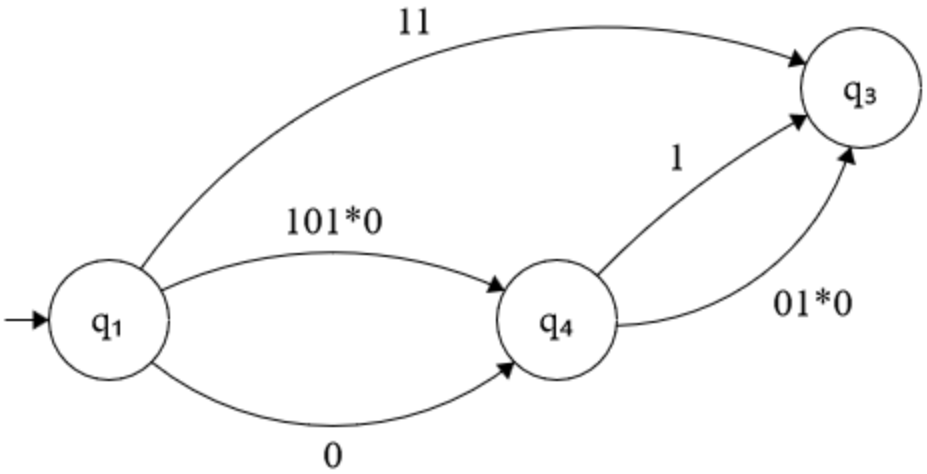
สร้าง G' ที่สอดคล้องกันโดยการเอา q_{rip} ออก

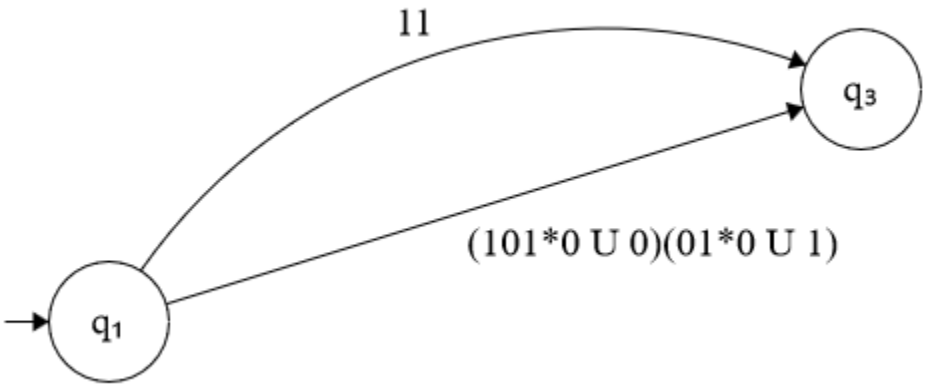
ทำซ้ำ

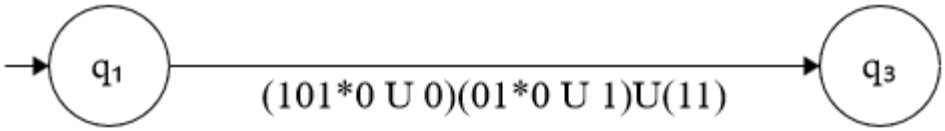
ลองแปลง




q7
q3
1
1
1
01*0
q1
q4
01*0
0


11
q3
1
101*0
q1
q4
01*0
0


11
q3
(101*0 U 0)(01*0 U 1)
q1


q1
(101*0 U 0)(01*0 U 1)U(11)
q3

Definition [GNFA]

A **generalized nondeterministic finite automata** is a 5-tuple, $(Q, \Sigma, \delta, q_{start}, q_{accept})$ where

- Q is the finite set of states
- Σ is the input alphabet
- $\delta: (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow \mathcal{R}$ is the transition function
- q_0 is the start state
- F is the accept state

ก่อนหน้านี้ transition function ของ DFA NFA จะเป็นฟังก์ชันที่รับ current state กับ input symbol แล้วได้ output ออกมาเป็น state ต่อไปหรือ set ของ state ที่เป็นไปได้

ใน DFA NFA เราสนใจว่า transition function บอกว่าเราจะไปที่ไหน

แต่ GNFA เราไม่ได้สนใจว่าจะทำงานอย่างไร เราสนใจแค่ว่าเราจะใช้มันในการยุบ

เวลาที่จะยุบ ผมติดกับใครแล้ว regular expression หน้าตาเป็นอย่างไร เพราะว่าจะได้เอาไปต่อได้ นั่นคือ เราสนใจแค่ว่า regular expression อะไรอยู่บนลูกศร

$$\delta: \left(Q - \{q_{accept}\}\right) \times \left(Q - \{q_{start}\}\right) \rightarrow \mathcal{R}$$

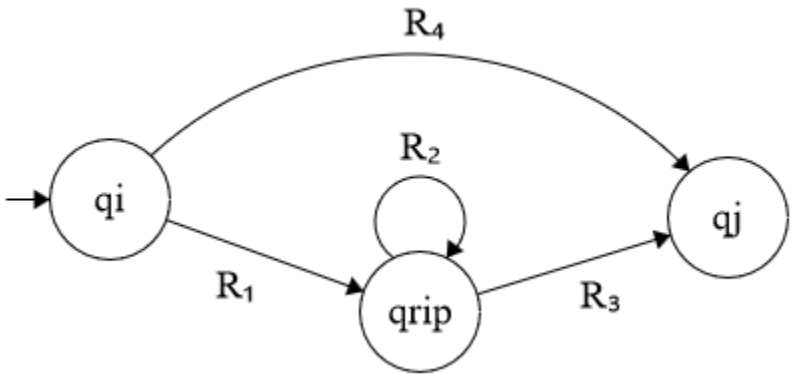
ทุกคู่ที่ตัวหน้าไม่ใช่ accept ตัวหลังไม่ใช่ start

accept ไม่ชี้ไปหาใคร

start ไม่มีใครชี้ไปหามัน

ทุกๆ คู่ regular expression อะไรอยู่บนตัวมัน

การลบ q_{rip}



Q' และ δ'

จาก $G = (Q, \Sigma, \delta, q_{start}, q_{accept})$ เราจะสร้าง equivalent $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$

เลือก q_{rip} ให้ $Q' = Q - \{q_{rip}\}$

สำหรับ $q_i \in Q' - \{q_{accept}\}$ และ $q_j \in Q' - \{q_{start}\}$

ให้

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_1)$$

เมื่อ $R_1 = \delta(q_i, q_{rip})$ $R_2 = \delta(q_{rip}, q_{rip})$ $R_3 = \delta(q_{rip}, q_j)$ $R_4 = \delta(q_i, q_j)$